МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут неперервної освіти

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Савченко А.С.

«___»_____2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТРА"

ЗА СПЕЦІАЛІЗАЦІЄЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ (ЗА ГАЛУЗЯМИ)"

Тема: «Автоматизація інформаційного порталу нерухомості»

Виконавець: Ярий Олексій Андрійович

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер: Райчев І. Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут неперервної освіти

Кафедра комп'ютерних інформаційних технологій

Галузь знань 12 «Інформаційні технології»

Спеціальність 122 «Комп'ютерні науки»

Спеціалізація «Інформаційні управляючі системи та технології (за галузями)»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Савченко А.С.
" " _____2019р.

З А В Д А Н Н Я

на виконання дипломної роботи студента

Ярого Олексія Андрійовича
(прізвище, ім'я, по батькові випускника в родинному відмінку)

1. Тема проекту (роботи): «Автоматизація інформаційного порталу нерухомості» затверджена наказом ректора від 22.11.2019р. № 2701/ст

2. Термін виконання проекту (роботи): з 25.11.2019 р. по 20.02.2020 р.

3. Вихідні дані до проекту (роботи): Інформаційний портал нерухомості з автоматичним розміщенням оголошень.

4. Зміст пояснювальної записки: вступ, особливості автоматизації інформаційних порталів, методи і засоби збору інформації з веб-сайтів, аналіз роботи веб-сайту, реалізація автоматизованої системи збору даних і їх публікація.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу: структурно-логічна модель автоматизованої системи, схема роботи сервісу збору інформації.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1	Виконати детальний аналіз літератури та Інтернет джерел за темою дипломного проекту. Розробити план дипломного проекту.	25.11.2019 – 07.12.2019	
2	Затвердити план дипломного проекту. Написати та затвердити вступ до дипломного проекту Провести консультації з науковим керівником, щодо першого розділу.	08.12.2019– 12.12.2019	
3	Розробка розділу 1: Особливості автоматизації інформаційних порталів.	13.12.2019 – 22.12.2019	
4	Розробка розділу 2: Методи і засоби побудови сервісу збору данних.	23.12.2019– 03.01.2020	
5	Розробка розділу 3: Реалізація розроблених моделей і методів на прикладі реального інформаційного порталу.	04.01.2020 – 08.01.2020	
6	Написати висновки до виконаного дипломного проекту.Оформити пояснювальну записку. Підготовити доповідь та презентацію.	09.01.2020– 29.01.2020	
7	Підписати необхідні документи у встановленому порядку. Підготовитися до захисту дипломного проекту.	30.01.2020 – 20.02.2020	

6. Дата видачі завдання: 25 листопада 2019 р.

Керівник дипломної роботи _____ Холявкіна Т.В.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Ярий О.А.

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Автоматизація інформаційного порталу нерухомості» викладена на сторінках. Містить малюнка, таблиць, джерел, додатка.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, ВЕБ-САЙТ ОГОЛОШЕНЬ, СЕРВІС ЗБОРУ ІНФОРМАЦІЇ.

Мета роботи: **Метою роботи** є створення сервісу збору та обробки інформації, моделей та алгоритмів для аналізу роботи веб-сайтів, побудованої із застосуванням сервісу та інформаційного порталу нерухомості.

Наукова новизна одержаних результатів полягає у виборі об'єкта дослідження – створення гнучкої системи збору даних з різних веб-сайтів.

Практична цінність результатів дипломної роботи полягає в розробці комплексу математичних моделей, алгоритмічного та програмного забезпечення для аналізу, збору та зберігання інформації для подальшого відображення на інформаційному порталі нерухомості.

Для вирішення поставлених завдань в роботі використовувалися такі методи:

1) аналіз та емуляція запитів;

2) проксінг запитів;

3) регулярні вирази;

4) синтаксичний аналіз.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HTTP — протокол передачі даних

CURL — проект програмного засобу;

URL — стандартизована адреса певного ресурсу;

Proxy — проксі-сервер;

JSON — текстовий формат обміну даними;

БД — база даних;

XML – розширювана мова розмітки;

PHP – гіпертекстовий препроцесор;

CRON — утиліта в операційних системах.

ВСТУП

Сучасні інформаційні портали нерухомості є важливою складовою частиною ринку нерухомості і їх автоматизація спрощує роботу порталу та їх користувачів.

Швидкий розвиток IP-мереж (насамперед Інтернет) породив нову тенденцію – використання веб-сайтів для розміщення оголошень з продажу та оренди нерухомості.

Однак таких рішень багато і через це пошук нерухомості ускладнюється, а саме – користувачі, які розміщують оголошення, повинні самостійно знаходити популярні інформаційні портали та розміщувати та інформацію. Крім того, для користувачів, які шукають житло, тепер необхідно аналізувати інформацію з декількох веб-сайтів та фільтрувати її, на наявність дублікатів та за особистими критеріями.

Відмінністю данного інформаційного порталу в автоматизації, сервіс буде автоматично збирати інформацію з веб-сайтів, аналізувати та фільтрувати її і розміщувати на веб-сайті. Користувачам, які розміщують оголошення, тепер не потрібно це робити на декількох платформах, а тим користувачам, які шукають нерухомість, не потрібно використовувати декілька платформ.

Метою роботи є створення моделей та алгоритмів для аналізу роботи веб-сайтів з оголошеннями нерухомості, побудова сервісу з обробки і публікації оголошень.

Для досягнення поставленої мети в роботі сформульовані і вирішені наступні завдання:

1) проведено аналіз моделі оголошення, дані які містить оголошення;

2) проведено аналіз роботи веб-сайтів, які будуть оброблятися сервісом;

3) проведено аналіз методів автоматизації порталів;

4) розроблена гнучкої структури сервісу, яка дозволяє обробляти інформацію з будь-яких веб-сайтів;

5) проведені експериментальні дослідження для оцінки якості структури сервісу і можливості їх застосування при вирішенні практичних завдань.

Реалізуються:

- по результатам аналізу роботи веб-сайтів створюється гнучкий інтерфейс, котрий описує роботу кожного порталу;

- моделі веб-сайтів, реалізуючих описаний інтерфейс, для збору інформації;

- модель оголошення, яка містить всю необхідну інформацію о нерухомості;

- моделі автоматизованого збору інформації

Практична цінність результатів дипломної роботи полягає в розробці комплексу математичних моделей, алгоритмічного та програмного забезпечення для аналізу, збору та зберігання інформації для подальшого відображення на інформаційному порталі нерухомості.

Достовірність та **обґрунтованість результатів**, отриманих в дипломної роботи, забезпечується відповідністю розроблених моделей та алгоритмів відомим теоретичним результатам і реальним процесом роботи інформаційного порталу та сервісу, котрий проводить збір та аналіз даних.

РОЗДІЛ 1

Особливості автоматизації інформаційних порталів

1.1. Поняття інформаційного порталу

Інтернет-портал (веб-портал, інформаційний портал) — сукупність поєднаних безпосередньо та через мережу "Інтернет" апаратних засобів, що включають комп'ютери та машинозчитувані електронні носії інформації із заздалегідь записаною на них інформацією та/або виконані з можливістю запису та зчитування інформації у вигляді комп'ютерних програм, баз даних і т. п., виконана з можливістю обробки зазначеної інформації та команд користувача системи та надання йому Інтернет-сервісів як результатів обробки зазначеної інформації і команд.[1]

Інтернет-портал для користувачів — сайт, що надає користувачеві Інтернету різні інтерактивні сервіси (Інтернет-сервіси), які працюють у рамках єдиного сайту.

Також портали працюють як точки доступу до інформації у Інтернеті або сайти, що допомагають користувачам у пошуку потрібної інформації через Інтернет. Такі портали представляють інформацію з різних джерел або тем об'єднаним способом і також називають навігаційними сайтами.

Всі портали виконують функції пошуку, а також, надають Інтернет-сервіси, наприклад: електронна по шта, стрічка новин тощо.

Ідея роботи порталу — створення або представлення критичної (найбільшої) маси Інтернет-сервісів, якими б можна було залучити до себе таку кількість користувачів-відвідувачів, яка буде постійно поповнюватися та збільшуватися.

Класифікація по спеціалізації інформації:

- Горизонтальний портал (або універсальний портал, портал загального характеру) — це портал, що охоплює багато тематик, представляє

набір сервісів (які обслуговують, по можливості, всі теми) й орієнтований на максимально широку аудиторію, на максимальне охоплення її інтересів. Найбільш відомі горизонтальні портали (за секторами Інтернету) — Yahoo! (англомовний), Mail.ru (російськомовний), Ukr.net, Яндекс (обидва — російськомовні та україномовні). Такі портали, як правило, сполучають у собі різноманітні функції, пропонують різноплановий вміст (контент) і різноманітні сервіси (новин, фінансові, розважальні, ігрові тощо).

- Вертикальний портал — це портал вузької тематичної спрямованості, що надає різні сервіси для користувачів мережі по певних інтересах і орієнтований на повне охоплення тематики або галузі діяльності. Якщо тематика вертикального порталу досить цікава, довкола нього може скластися так звана «спільнота» (community) — більш-менш постійна група осіб, що, наприклад, систематично спілкуються між собою в чаті такого порталу.
- Змішаний портал — портал, що сполучає в собі функції електронної торгівлі та класичні довідкові сервіси. Прикладами таких порталів є інтернет-магазини. Також змішаними порталами вважаються такі вертикальні портали, які починають займатися бізнесом у своїх спеціалізованих розділах.

1.2. Особливості функціонування інформаційних порталів

Функціонування інформаційного порталу залежить від його типу, а саме правам доступу користувачам на веб-сайті, найпопулярніші:

відкритий портал - ресурс, на якому усі користувачі можуть вільно розміщувати інформацію, це можуть бути як новини так і оголошення. На даний момент кількість таких порталів зменшується через появу модерацію.

частково-відкриті портали - ресурси, на яких усі користувачі можуть подати запит на публікацію матеріалу, але він буде розміщений тільки після підтвердження адміністратором веб-сайту. Більшість порталів належать до даного типу, наприклад - olx.ua, wikipedia.org, prom.ua.

закриті портали - веб-сайти, розміщенням інформації на яких займаються лише адміністратори та модератори ресурсу, частіше за все це портали з новинами.

автоматизовані портали - портали, розміщенням інформації на яких займаються бот-програми або сервіси по збору інформації.

Основним функціоналом інформаційного порталу є відображення інформації, це може бути як звичайний текстовий пост або новина, так і оголошення або об’ява.

1.3. Веб-сайт та веб-сервіс

Веб-сайт, або сайт (англ. website, від web (веб) і site (місце)) — сукупність веб-сторінок, доступних у мережі Інтернет, які об'єднані як за змістом, так і за навігацією під єдиним доменним ім'ям. Фізично сайт може розміщуватися як на одному, так і на кількох серверах.[2]

Сайтом також називають вузол мережі Інтернет, комп'ютер, за яким закріплена унікальна IP-адреса, і взагалі будь-який об'єкт в Інтернеті, за яким закріплена адреса, що ідентифікує його в мережі (FTP-site, WWW-site тощо).

Набір зв'язаних між собою інформаційних онлайнових ресурсів, призначених для перегляду через комп'ютерну мережу за допомогою спеціальних програм — браузерів. Веб-вузол може бути набором документів в електронному вигляді, онлайн службою.

Веб-сервіс (англ. web service) — програмна система, що ідентифікується URI, і публічні інтерфейси та прив'язки якої визначені та описані мовою XML. Опис цієї програмної системи може бути знайдено іншими програмними системами, які можуть взаємодіяти з нею відповідно до цього опису з використанням повідомлень, що базуються на XML або JSON та передаються за допомогою інтернет-протоколів.[3]

Термін "вебслужба" введено організацією W3C і застосовується до багатьох різних систем, але в основному термін стосується клієнтів та серверів,

що взаємодіють за допомогою повідомлень протоколу SOAP. В обох випадках припускається, що існує також опис доступних операцій у форматі WSDL. Хоча наявність цього опису не є вимогою SOAP, а радше передумовою для автоматичного генерування коду на платформах Java та .NET на стороні клієнта.

Тобто, веб-сайт відрізняється від веб-сервісу наявністю візуального оформлення та користувачами, у випадку сайту це люди, а у сервісу - програмне забезпечення.

1.4. Взаємодія веб-сервісів

Веб-сервіси взаємодіють через веб-протоколи, найпопулярніші - HTTP та WebSocket.

HTTP — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів.[4]

HTTP належить до протоколів моделі OSI 7-го прикладного рівня.

Основним призначенням протоколу HTTP є передача веб-сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються і інші файли, які пов'язані з веб-сторінками (зображення і застосунки), так і не пов'язані з ними (у цьому HTTP конкурує з складнішим FTP).

HTTP припускає, що клієнтська програма — веб-браузер — здатна відображати гіпертекстові веб-сторінки та файли інших типів у зручній для користувача формі. Для правильного відображення HTTP дозволяє клієнтові дізнатися мову та кодування символів веб-сторінки й/або запитати версію сторінки в потрібних мові/кодуванні, використовуючи позначення із стандарту MIME.

Якщо в URL зі схемою http:// не вказаний порт, то за замовчуванням береться 80, (для схеми https — 443).

HTTP — протокол прикладного рівня, схожими на нього є FTP і SMTP. Обмін повідомленнями йде за звичайною схемою «запит-відповідь». Для ідентифікації ресурсів HTTP використовує глобальні URI. На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами «запит-відповідь». Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації про стан, пов'язаний з останніми запитами та відповідями. Браузер, котрий відправляє запити, може відстежувати затримки відповідей. Сервер може зберігати IP-адреси та заголовки запитів останніх клієнтів. Проте, згідно з протоколом, клієнт та сервер не мають бути обізнаними з попередніми запитами та відповідями, у протоколі не передбачена внутрішня підтримка стану й він не ставить таких вимог до клієнта та сервера.

Кожен запит/відповідь складається з трьох частин:

- стартовий рядок;
- заголовки;
- тіло повідомлення, що містить дані запиту, запитаний ресурс або опис проблеми, якщо запит не виконано.

Обов'язковим мінімумом запиту є стартовий рядок. Починаючи з HTTP/1.1 обов'язковим став заголовок Host: (щоб розрізнити кілька доменів, які мають одну й ту ж IP-адресу).

WebSocket — це протокол, що призначений для обміну інформацією між браузером і веб-сервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. WebSocket спроектовано для втілення у веб-браузерах та веб-серверах, але може також використовуватись будь-яким клієнт-серверним застосунком. Прикладний програмний інтерфейс WebSocket був стандартизований W3C, крім того протокол WebSocket стандартизований IETF як RFC 6455.[5]

У веб-застосунках доцільно використовувати протокол за необхідності відображення інформації в real-time. Альтернативна технологія — Server-sent events.

1.5. Синтаксичний аналіз веб-сайту

Синтакси́чний ана́ліз (па́рсинг) (англ. parsing) — в інформатиці це процес аналізу вхідної послідовності символів, з метою розбору граматичної структури згідно із заданою формальною граматикою. Синтаксичний аналізатор (англ. parser) — це програма або частина програми, яка виконує синтаксичний аналіз. [6]

Синтаксичний аналізатор — це програмний компонент, який приймає вхідні дані (часто текст) і створює структуру даних — часто дерево розбору, абстрактне дерево синтаксису або іншу ієрархічну структуру — забезпечує структурне представлення вводу, перевіряє правильність синтаксису в процесі. Для аналізу можуть передувати або слідувати інші кроки, або їх можна об'єднати в один крок. Аналізатору часто передує окремий лексичний аналізатор, який створює токени з послідовності введених символів; Крім того, їх можна об'єднати у парсинг без сканування. Аналізатори можуть бути запрограмовані вручну або автоматично або напів автоматично генератором парсерів. Розбір допомагає шаблону, який виробляє відформатований вихід. Вони можуть використовуватись у різних ділянках, але часто з'являються разом, наприклад, пара scanf/printf, або як вхідний (аналіз вхідних даних) та вихідний етапи (створення кінцевого коду) компілятора.

Вхідними даними для синтаксичного аналізатора часто є текст на деякій комп'ютерній мові, але також може бути текстом природною мовою або менш структурованими текстовими даними, в цьому випадку, як правило, витягуються лише окремі частини тексту, а не дерево розбору. Параметри відрізняються від дуже простих функцій, таких як scanf, до складних програм, таких як інтерфейс компілятора C++ або HTML-аналізатор веб-браузера.

Важливий клас простий синтаксичний аналіз виконується за допомогою регулярних виразів, в яких група регулярних виразів визначає регулярну мову та двигун регулярного виразу, автоматично генеруючи аналізатор для цієї мови, що дозволяє узгодити шаблон та вилучення тексту. В інших контекстах регулярні вирази замість цього використовуються перед розбором, як етап лексизації, вихід якого потім використовується аналізатором.

Використання аналізаторів залежить від вхідних даних. У випадку з мовами даних часто використовується синтаксичний аналізатор як функція читання файлів у програмі, наприклад, читання в HTML або XML-тексті; ці приклади є мовами розмітки даних. У випадку мов програмування є компонентом компілятора або інтерпретатора, який аналізує початковий код мови комп'ютерного програмування для створення певної форми внутрішнього представлення; аналізатор є ключовим кроком в інтерфейсі компілятора. Мови програмування, як правило, вказуються в термінах детерміністичної контекстно-вільної граматики, оскільки для них можуть бути написані швидкі та ефективні аналізатори. Для компіляторів сам аналіз може бути виконаний за один прохід або кілька проходів.

Майбутні недоліки компілятора з одним прохідним процесом у значній мірі можуть бути вирішені шляхом додавання виправлень, коли передбачається виправлення впродовж прямого переходу, а виправлення застосовуються в зворотньому напрямку, коли поточний сегмент програми є таким, що має бути завершений. Приклад, коли такий механізм виправлення може бути корисним, буде формальним твердженням GOTO, де ціль GOTO невідома, доки не буде пройдено сегмент програми. У такому випадку застосування виправлення буде відкладено, доки не буде визначено куди вказує GOTO. Очевидно, що відсталий GOTO не вимагає виправлення.

Контекстні граматики обмежені в тій мірі, в якій вони можуть виразити всі вимоги до мови. Неформально, причиною є те, що пам'ять в такій мові обмежена. Граматика не запам'ятовує наявність конструкції над довільним

введенням; це необхідно для мови, в якій, наприклад, ім'я повинно бути оголошено, перш ніж може бути посилання на нього. Однак більш потужні граматики, які можуть обійти це обмеження, не можуть бути ефективно розібрані. Таким чином, загальною стратегією є створення аналізатора для контекстно-вільної граматики, який приймає потрібні конструкції мови (тобто він приймає деякі недійсні конструкції); пізніше, небажані конструкції можуть бути відфільтровані на етапі семантичного аналізу (контекстного аналізу).

Поверхне́во-синтаксѝчний ана́ліз (англ. shallow parsing, також англ. chunking, «light parsing») — це аналіз речення, який спершу ідентифікує складові частини речень (іменники, дієслова, прикметники тощо), а потім пов'язує їх в одиниці вищого порядку, які мають окремі граматичні значення (іменникові групи або фрази, дієслівні групи тощо). І хоча найелементарніші алгоритми поверхнево-синтаксичного аналізу просто пов'язують складові частини на основі елементарних шаблонів пошуку (наприклад, заданих регулярними виразами), підходи, які застосовують методи машинного навчання (класифікатори, тематичне моделювання тощо) можуть враховувати контекстну інформацію, і відтак формувати фрагменти таким чином, щоби вони краще відображували семантичні зв'язки між основними складовими. Тобто, ці досконаліші методи обходять ту проблему, що поєднання елементарних складових можуть мати різні значення вищого рівня залежно від контексту речення.

Він є методикою, широко вживаною в обробці природної мови. Він є подібним до поняття лексичного аналізу для комп'ютерних мов. Під назвою «гіпотеза поверхневої структури» (англ. Shallow Structure Hypothesis) його також використовують, щоби пояснювати, чому людям, які вивчають другу мову, часто не вдається правильно розбирати складні речення.

ВИСНОВОК ДО РОЗДІЛУ 1

Приймаючи до уваги розвиток інтернет технологій можна помітити що кількість ресурсів не завжди позитивно впливає на користувачів.

Різноманітність інформаційних порталів збільшується і саме тому, користувачу необхідно слідкувати за трендами та популярністю використовуваних ресурсів. Також тепер користувачеві потрібно слідкувати за дублюванням оголошень, що також негативно впливає на зручність використання порталів. Тому вирішено зробити автоматизовану систему, котра буде аналізувати та збирати інформацію оголошень та представляти у зручному вигляді для користувача.

РОЗДІЛ 2

АНАЛІЗ МЕТОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ АВТОМАТИЗОВАНОГО ІНФОРМАЦІЙНОГО ПОРТАЛУ

2.1. Вибір інструментальних засобів побудови системи

Сьогодні для розробки автоматизованих інформаційного порталу пропонується широкий спектр інструментальних засобів. Від вибору середовища розробки залежить багато, які впливають на систему. В якості структурного було вирішено розділи портал на дві відокремлені частини, а саме: веб-сайт(на ньому будуть відображатися оголошення нерухомості) та веб-сервіс(який буде проводити збір та аналіз даних з веб-сайтів).

Для розробки веб-сайту будемо використовувати CMS систему, а саме WordPress, так як він дає можливість гнучко керувати контентом на сайті.

Веб-сервіс буде розроблений повністю власноруч, але з використанням допоміжних сервісів та програмного забезпечення. Вер-сервіс розроблен на скриптовій мові програмування PHP, з використанням розширення:

- *mysqli* — надає можливість відправляти запити до бази-данних;
- *libcurl* — розширення для відправки HTTP запитів;

Аналіз роботи сервісу проводиться завдяки інтеграції зі стороннім сервісом Sentry, особливістю котрого є можливість сповіщення та перегляду помилок на веб-сайті сервісу.

Гео-кодування та перевірка валідності адреси нерухомості проводиться через веб-сервіс Google Maps API. Перевірка на валідність необхідна для впевненості у правильності даних, які вказав власник оголошення. Сервіс гео-кодування необхідний для відображення нерухомості на карті.

Серед необхідного програмного забезпечення слід відзначити лише веб-сервер, використовуємо *NGINX* та сервіс з базою даних *MYSQL*.

2.1.1. ДОКУМЕНТАЦІЯ ПО РОБОТІ CURL В PHP

У PHP включена підтримка libcurl - бібліотека функцій, написаної Деніелом Стенберг (Daniel Stenberg), яка дозволяє взаємодіяти з безліччю різних серверів за багатьма різних протоколах. В даний час libcurl підтримує протоколи http, https, ftp, gopher, telnet, dict, file і ldap. libcurl також вміє працювати з сертифікатами HTTPS, посилати запити до HTTP-серверів методами POST і PUT, завантажувати файли по протоколах HTTP і FTP (останнє можна зробити за допомогою модуля FTP), використовувати проксі-сервери, cookies та аутентифікацію користувачів.[7]

Методи та функції необхідні для відправки HTTP запиту:

- **curl_init** - Ініціалізує сеанс cURL.

curl_init ([string $url = NULL]) : resource

Параметр url, якщо вказано, встановлює адресу сайту або сервісу

- **curl_setopt** - Встановлює параметр для сеансу CURL

curl_setopt (resource $ch , int $option , mixed $value) : bool

Параметри — *ch* (Дескриптор cURL, отриманий з curl_init ()), *option (в*становлюваний параметр CURLOPT_XXX) та *value (з*начення параметра option)

- **curl_exec** - Виконує запит cURL

curl_exec (resource $ch) : mixed

Параметри — *ch* (Дескриптор cURL, отриманий з curl_init ())

Повертає результат запиту.

- **curl_close** - Завершує сеанс cURL

curl_close (resource $ch) : void

Параметри — *ch* (Дескриптор cURL, отриманий з curl_init ())

Зумовлені константи необхідні для роботи зі сторонніми сервісами:

- CURLOPT_URL — URL адрсса веб-сервісу;

- CURLOPT_RETURNTRANSFER — *TRUE* для повернення результату передачі в якості рядка з curl_exec () замість прямого виведення в браузер;
- CURLOPT_ENCODING — Вміст заголовка "Accept-Encoding:". Це дозволяє декодувати запит;
- CURLOPT_HTTP_VERSION — версія протоколу HTTP;
- CURLOPT_POSTFIELDS — Всі дані, що передаються в HTTP POST-запиті;
- CURLOPT_HTTPHEADER — Масив встановлюються HTTP-заголовків, в форматі *array ('Content-type: text / plain', 'Content-length: 100');*
- CURLOPT_PROXY — HTTP-проксі, через який будуть направлятися запити;
- CURLOPT_PROXYTYPE — Тип використовуваного проксі протоколу

2.1.2. ДОКУМЕНТАЦІЯ ПО РОБОТІ MYSQL В PHP

Розширення Об'єкти даних PHP (PDO) визначає простий і узгоджений інтерфейс для доступу до баз даних в PHP. Кожен драйвер бази даних, в якому реалізований цей інтерфейс, може представить специфічні для бази даних функціонал у вигляді стандартних функцій розширення. Але треба зауважити, що саме по собі розширення PDO не дозволяє маніпулювати доступом до бази даних. Щоб скористатися можливостями PDO, необхідно використовувати Відповідний конкретної базі даних PDO драйвер.[8]

PDO забезпечує абстракцію доступу до даних, це означає, що незалежно від того, яка конкретна база даних використовується, ви можете користуватися одними і тими ж функціями для виконання запитів і вибірки даних. PDO НЕ абстрагує саму базу даних, це розширення НЕ переписує SQL-запити і НЕ

емулює відсутній в СУБД функціонал. Якщо потрібно саме це, необхідно скористатися повноцінною абстракцією бази даних.

Розширення PDO впроваджені в PHP 5.1, але також є в 5.0 у вигляді PECL-розширення; PDO вживає новий OO-функціонал з ядра PHP 5 і тому Чи не буде працювати з Раннє версіями PHP.

Підключення і управління підключеннями

З'єднання встановлюються автоматично при створенні об'єкта PDO від його базового класу. Не має значення, який драйвер ви хочете використовувати; ви завжди використовуєте ім'я базового класу. Конструктор класу приймає аргументи для завдання джерела даних (DSN), а також необов'язкові ім'я користувача і пароль (якщо є).

У разі помилки при підключенні буде викинуто виключення PDOException. Його можна перехопити і обробити, або залишити на відкуп глобальному оброблювачу помилок, який ви задали функцією *set_exception_handler()*.

Транзакції і автоматична фіксація змін

Тепер знаємо, як підключатися до баз даних за допомогою PDO. Але перед тим як виконувати запити, вам необхідно зрозуміти, як PDO управляє транзакціями. Якщо ви раніше не стикалися з транзакціями, вони володіють чотирма головними властивостями, це Атомарність, Узгодженість, Ізольованість і Довговічність (ACID). Говорячи простою мовою, будь-які дії, вчинені в рамках транзакції, гарантовано будуть виконані безпечно для бази даних, і на них не вплинуть інші підключення до цієї бази, навіть якщо ці дії відбуваються в кілька етапів. Транзакційні операції можна скасовувати за запитом (якщо транзакція ще не зафіксована), що спрощує обробку помилок в скриптах.

Механізм транзакцій реалізований шляхом "тимчасового збереження" всіх змін і подальшого застосування цих змін, як єдиного цілого. Це дозволяє домогтися різкого збільшення ефективності подібних змін. Іншими словами,

транзакції можуть зробити ваші скрипти швидшими і потенційно більш стабільними (але для цього необхідно коректно використовувати цей механізм).

На жаль, не всі бази даних підтримують транзакції, тому PDO коли Ви створюєте з'єднання працює в режимі так званої "автоматичної фіксації". Режим автофіксації означає, що кожен запит до бази даних, який ви виконуєте, неявно полягає в транзакцію, якщо СУБД їх підтримує. Якщо база даних не підтримує цей механізм, запит обробляється без транзакції. Щоб явно позначити початок транзакції, ви повинні використовувати метод PDO :: beginTransaction (). Якщо драйвер не підтримує механізм транзакцій, буде викинуто виключення PDOException (незалежно від обраного способу обробки помилок: подібні ситуації - це завжди серйозна недоробка). Після того, як ви зробите транзакцію, ви можете зафіксувати її методом PDO :: commit () або відкотити методом PDO :: rollBack (), в залежності від того, успішно виконано ваш код всередині транзакції чи ні.

При завершенні роботи скрипта або при закритті з'єднання, PDO автоматично відкочується все незавершені транзакції. Це робиться, щоб запобігти порушенням цілісності бази даних у випадках, коли скрипт несподівано перериває роботу. Якщо ви явно не зафіксували зміни, передбачається, що щось пішло не так. Тому відкат змін - найбільш безпечний вихід із ситуації.

Підготовлені запити і процедури

Більшість баз даних підтримують концепцію підготовлених запитів. Що це таке? Це можна описати, як якийсь вид скомпільованої шаблону SQL запиту, який буде запускатися додатком і налаштовуватися за допомогою вхідних параметрів. У підготовлених запитів має дві незаперечних переваги:

- Запит необхідно один раз підготувати і потім його можна запускати стільки раз, скільки потрібно, причому як з тими ж, так і з відмінними параметрами. Коли запит підготовлений, СУБД аналізує його, компілює і оптимізує план його виконання. У разі складних запитів цей процес може

займати відчутний час і помітно уповільнити роботу програми, якщо буде потрібно багато разів виконувати запит з різними параметрами. При використанні підготовленого запиту СУБД аналізує / компілює / оптимізує запит будь-якої складності тільки один раз, а додаток запускає на виконання вже підготовлений шаблон. Таким чином підготовлені запити споживають менше ресурсів і працюють швидше.

- Параметри підготовленого запиту не потрібно екранувати лапками; драйвер це робить автоматично. Якщо в додатку використовуються виключно підготовлені запити, розробник може бути впевнений, що ніяких SQL-ін'єкцій трапитися не може (однак, якщо інші частини тексту запиту створюються з не екранованим введенням, то SQL ін'єкція як і раніше можлива).

Підготовлені запити також корисні тим, що PDO може емулювати їх, якщо драйвер бази даних немає подібного функціоналу. Це означає, що додаток може користуватися однією і тією ж методикою доступу до даних незалежно від можливостей СУБД.

2.1.3. ІНТЕГРАЦІЯ З GOOGLE MAPS API

GOOGLE MAPS API PLACES - це служба, яка повертає інформацію про місця, що використовують HTTP-запити. Місця визначені в цьому API як заклади, географічні місця розташування або визначні визначні місця.[9]

Доступні такі запити на місце:

- *Place Search* повертає список місць на основі місцезнаходження користувача або рядка пошуку.
- *Place Details* повертає більш детальну інформацію про певне місце, включаючи відгуки користувачів.
- *Place Photos* надає доступ до мільйонів фотографій, пов’язаних із місцем, що зберігаються в базі даних Google Place.
- *Place Autocomplete* автоматично заповнює ім’я та / або адресу місця за типом користувачів.

- *Query Autocomplete* надає послугу передбачення запитів для текстових географічних пошукових запитів, повертаючи запропоновані запити як тип користувачів.

До кожної з служб звертається як HTTP-запит і повертає або відповідь JSON або XML. Усі запити до служби Місця на карті повинні використовувати протокол https: // і містити ключ API.

API Місця на карті використовує ідентифікатор місця для унікальної ідентифікації місця. Докладніше про формат та використання цього ідентифікатора в API Місця та інших API дивіться в документації Ідентифікаторів місць.

Серед можливих методів нам необхідний лише *Place Search* та *Place Details*.

API *Place Details* дозволяє шукати інформацію про місця, використовуючи різні категорії, включаючи заклади, визначні місця та географічні місця. Ви можете шукати місця поблизу чи текстового рядка. Пошук місць повертає список місць разом із зведеною інформацією про кожне місце; додаткову інформацію можна отримати за запитом "Інформація про місце".

Запит на пошук місця займає введення тексту та повертає місце. Вхідними даними можуть бути будь-які текстові дані Місця, такі як ім’я, адреса чи номер телефону. Запит повинен бути рядком. Запит "Знайти місце", використовуючи не рядові дані, такі як координати lat / lng або плюс, породжує помилку.

Запит на пошук місця - це HTTP-адреса такої форми:

https://maps.googleapis.com/maps/api/place/findplacefromtext/output?parameters

де *output* може бути одним із наступних значень:

- json (рекомендовано) вказує на вихід у JavaScript Object Notation
- xml позначає вихід як XML

Для ініціювання запиту на пошук місця потрібні певні параметри. Як це прийнято в URL-адресах, всі параметри розділені за допомогою символу ampersand (&).

Необхідні параметри

key - ключ API вашої програми. Цей ключ ідентифікує вашу програму. Див. Розділ Отримання ключа для отримання додаткової інформації.

input - введення тексту, що ідентифікує ціль пошуку, наприклад ім'я, адресу чи номер телефону. Вхід повинен бути рядком. Не рядкове введення, такий як координата lat / lng або плюс, породжує помилку.

inputtype - Тип вводу. Це може бути один із текстових запитів або номер телефону. Номери телефонів повинні бути в міжнародному форматі (з префіксом знака плюс ("+"), після чого код країни, потім сам номер телефону). Див. Рекомендацію MCE E.164 для отримання додаткової інформації.

Необов’язкові параметри

language - код мови із зазначенням, на якій мові результати, якщо це можливо, слід повернути. Пошукові запити також є упередженими до вибраної мови; результати на вибраній мові можуть бути отримані більш високим рейтингом. Перегляньте список підтримуваних мов та їх коди. Зауважте, що ми часто оновлюємо підтримувані мови, тому цей список може бути не вичерпним.

fields - поля, що визначають типи даних про місця, які потрібно повернути, розділені комою.

locationbias - віддайте перевагу результатам у визначеній області, вказавши або радіус плюс lat / lng, або дві пари lat / lng, що представляють точки прямокутника. Якщо цей параметр не вказаний, API використовує збиток IP-адреси за замовчуванням.

- IP-зміщення: доручає API використовувати зрушення IP-адреси. Передайте рядок ipbias (ця опція не має додаткових параметрів).
- Точка: Одинарна координата lat / lng. Використовуйте наступний формат:

- точка: лат, lng.
- Кругова: рядок із зазначенням радіуса в метрах, плюс lat / lng в десяткових градусах. Використовуйте такий формат: коло: радіус @ lat, lng.
- Прямокутний: рядок із зазначенням двох пар / lng пар у десяткових градусах, що представляють точки південь / захід та північ / схід прямокутника. Використовуйте наступний формат: прямокутник: південь, захід | північ, схід. Зверніть увагу, що значення схід / захід приведені до діапазону -180, 180, а значення північ / південь зафіксовані до діапазону -90, 90.

Fields (Поля)

Використовуйте параметр *fields* для визначення списку розділених комами типів даних про місце для повернення. Наприклад: *fields=opening_hours,icon,geometry*. Використовуйте пряму косу рису, задаючи значення складових. Наприклад: геометрія / розташування.

Поля відповідають результатам пошуку місць на місцях і поділяються на три категорії рахунків: Основні, Контактні та Атмосферні. Основні поля тарифікуються за базовою ставкою і не вимагають додаткових витрат. Контактні та атмосферні поля тарифікуються за більш високою швидкістю. Додаткову інформацію див. У ціновій таблиці. Атрибути (*html_attributions*) завжди повертаються з кожним викликом, незалежно від того, чи було запитано поле.

Basic (Основні) - Основна категорія включає наступні поля: відформатована адреса, геометрія, значок, ім'я, постійно закриті, фотографії, місце код, плюс код, типи.

Contact (Контактна інформація) - Категорія контактів включає таке поле: opens_hours (Пошук місця повертається лише до open_now; використовуйте запит про деталі місця, щоб отримати повні результати відкриття_часу).

Atmosphere (Атмосфера) - Категорія «Атмосфера» включає такі поля: рівень ціни, рейтинг, користувальницькі оцінки тотал

Служба *Text Search* Google Places API - це веб-служба, яка повертає інформацію про набір місць на основі рядка - наприклад, "піца в Нью-Йорку" або "магазини взуття поблизу Оттави" або "123 Main Street". Служба відповідає у відповідь зі списком місць, що відповідають текстовому рядку та будь-яким встановленим зміщенням місцеположення.

Послуга особливо корисна для створення неоднозначних адресних запитів в автоматизованій системі, а не адресовані компоненти рядка можуть відповідати бізнесу, а також адресам. Прикладами неоднозначних адресних запитів є неповні адреси, неправильно відформатовані адреси або запит, що включає не адресовані компоненти, такі як назви підприємств.

Відповідь на пошук включатиме список місць. Ви можете надіслати запит щодо деталей місця, щоб отримати додаткову інформацію про будь-яке місце у відповіді.

Запит на пошук тексту - це HTTP-адреса такої форми:

https://maps.googleapis.com/maps/api/place/textsearch/output?parameters

Необхідні параметри:

- *query* - текстовий рядок, за яким слід шукати, наприклад: "ресторан" або "123 Головна вулиця". Послуга Google Місця на карті поверне збіги кандидатів на основі цього рядка та впорядковує результати, виходячи з їхньої визнаної релевантності. Цей параметр стає необов’язковим, якщо параметр типу також використовується в запиті пошуку.
- *key* - ключ API вашої програми. Цей ключ ідентифікує вашу програму. Див. Розділ Отримання ключа API для місць, щоб дізнатися, як створити проект API та отримати свій ключ.

Необов’язкові параметри:

- *region* - код регіону, вказаний у вигляді знаку ccTLD (домен верхнього рівня коду країни). Більшість кодів ccTLD ідентичні кодам ISO 3166-1, за деякими винятками. Цей параметр впливатиме на результати пошуку, але

не повністю обмежуючи їх. Якщо більш відповідні результати існують за межами зазначеного регіону, вони можуть бути включені. Коли цей параметр використовується, ім'я країни опускається з отриманого форматованого_адреса для отримання результатів у вказаному регіоні.

- *location* - широта / довгота, навколо якої можна отримати інформацію про місце. Це потрібно вказати як широту, довготу. Якщо ви вказуєте параметр розташування, ви також повинні вказати параметр радіуса.
- *radius* - визначає відстань (у метрах), в межах якого потрібно змістити результати. Максимально дозволений радіус - 50 000 метрів. Результати в цьому регіоні будуть ранжировані вище, ніж результати поза пошуковим колом; однак можуть бути включені помітні результати за межами радіусу пошуку.
- *language* - код мови із зазначенням, на якій мові результати, якщо це можливо, слід повернути. Перегляньте список підтримуваних мов та їх коди. Зауважте, що ми часто оновлюємо підтримувані мови, тому цей список може бути не вичерпним.
- *minprice* та *maxprice* (необов’язково) - Обмежує результати лише в тих місцях, що знаходяться в межах вказаного рівня цін. Дійсні значення знаходяться в діапазоні від 0 (найдоступніший) до 4 (найдорожчий) включно. Точна сума, вказана конкретним значенням, буде залежати від регіону до регіону.
- *opennow* - Повертає лише ті місця, які відкриті для бізнесу під час надсилання запиту. Місця, які не визначають години роботи в базі даних Google Місця, не повертаються, якщо ви включите цей параметр у свій запит.
- *pagetoken* - Повертає до 20 результатів із попередньо запущеного пошуку. Якщо встановити параметр pagetoken, буде здійснено пошук з тими ж самими параметрами, які використовувались раніше - всі параметри, крім pagetoken, будуть ігноровані.
- *type* - Обмежує результати місцями, що відповідають вказаному

типу. Може бути вказаний лише один тип (якщо передбачено більше одного типу, усі типи після першого запису ігноруються). Перегляньте список підтримуваних типів.

Ви можете змістити результати до вказаного кола, передаючи параметр розташування та радіус. Це доручить службі Google Місця на карті віддати перевагу показу результатів у цьому колі. Результати поза визначеною областю все ще можуть відображатися. Для покращення релевантності результатів для неоднозначних запитів рекомендується зміна результатів для регіону чи кола.

2.2. Алгоритм роботи веб-сервісу зі сторонніми веб-сайтами

Алгоритм роботи веб-сервісу складається з таких етапів:

1. Отримуємо модель конкретного сайту
2. В залежності від принципу роботи сайту отримуємо список усіх оголошень, одним запитом або декількома
3. Актуалізуємо отриманий список з оголошеннями в БД
4. Список нових оголошень, отриманих після актуалізації, перевіряємо на валідність та вносимо до БД

Алгоритм створений таким чином, щоб утримувати всі дані максимально актуальними, для цього на сервері встановлено *CRON,* це утиліта в операційних системах Unix та Linux, яка дозволяє користувачам виконувати команди або скрипти (групи команд) автоматично в заданий час. Завдяки даному програмному забезпеченню ми можемо гнучко регулювати період оновлення даних.

Розглянемо кожен етап окремо. Під час виконання першого етапу, ми повинні отримати ту модель класу, дані якої були оновлені останні, для цього записуємо час останньої обробки даних з кожного веб сайту. Отримавши модель конкретного веб-сайту перевіряємо його тип, в залежності від реалізованих методів класу моделі. Є лише два типи моделей, за методом отримання списку

оголошень: одним запитом(якщо відсутня пагінація) або N-кількістю запитів, де N кількість сторінок.

Процес актуалізації оголошень розподілен на декілька частин:

1. Отримуємо список уже внесених до БД оголошень, звіряємо їх за ідентифікатором(у кожно веб-сайту він різний) та актуалізуємо ціну за необхідністю

2. Отримуємо список оголошень, які є у БД але відсутні у списку з веб-сайту, ці оголошення вже не актуальні, тому їх видаляємо.

3. Відокремлюємо зі списку нові оголошення, яких ще не має у БД

4. Список нових оголошень, по черзі, розбираємо необхідні дані та вносимо до БД.

2.3. Алгоритм роботи веб-сервісу із інформаційним порталом

Так як, інформаційний портал використовує CMS, у нас є можливість використовувати її базовий функціонал, а саме — відображення даних з БД. Також, завдяки WordPress, ми маємо можливість використовувати *wp_cron*, це аналог *CRON* встановлений на сервері з веб-сервісом. Для роботи з ним нам необхідно викликати метод *wp_schedule_event — с*творює багаторазову крон-завдання. Встановлює хук, який буде викликатися кожен раз через вказаний інтервал часу.

wp_schedule_event($timestamp, $recurrence, $hook, $args);

Необхідні параметри для виклику функції:

- *$timestamp* (число) (обов'язковий) - Початкова мітка часу, з якої хук почне працювати. Потрібно вказувати в форматі UNIX (32165487). WP cron використовує час UTC / GMT, а не локальне. Використовуйте функцію *time()*, яка завжди в GMT.

- *$recurrence* (рядок) (обов'язковий) - Як часто має повторюватися дію. Допустимі значення нижче. Ви можете створити свій інтервал використовуючи фільтр cron_schedules з функції wp_get_schedules().

- hourly - щогодини;
- twicedaily - двічі в день;
- daily — щодня.
- *$hook* (рядок) (обов'язковий) - Назва хука, який потрібно виконувати.
- *$args* (масив) - Аргументи, які потрібно передати в виконуваний хук.

Тому алгоритм роботи буде такий:

1. Отримуємо актуальний список оголошень з усією інформацією
2. Інформацію о оголошеннях які вже є на сайті — оновлюємо
3. Оголошення, які розміщені на сайті але відсутні у списку — видаляємо
4. Нові оголошення вносимо до БД

Для отримання даних с веб-сервісу буде відправлений HTTP запит, завдяки *libcurl,* а дані будуть формуватися у *JSON.*

2.4. Аналіз роботи веб-сайту

Для проведення парсингу веб-сайту нам необхідно проаналізувати його роботу, а саме - метод отримання та відображення інформації на сторінці. Існує два основні методи: синхронне та асинхронне.

Синхронне відображення веб-сторінки полягає у завантаженні усієї інформації за один запит, без додаткової роботи *JavaScript.* Для парсингу таких веб-сайтів потрібно відправити HTTP запит на конкретну URL адресу та у відповідь отримати HTML, для подальшого парсингу.

Асинхронне відображення веб-сторінки проходить завдяки *JavaScript,* завдяки йому відправляються додаткові запити до серверу і після обробки вставляються в поточну сторінку. Запити відправляються за допомогою технології *AJAX*.

AJAX - це підхід до реалізації, що включає в себе клієнтський скрипт (запускається в браузері), який обмінюється даними з веб-сервером. AJAX - додаток для відправки даних може використовувати XML, простий текст або JSON. Але в цілому браузер використовує об'єкт XMLHttpRequest для запиту даних з сервера, а JavaScript - для відображення даних.[10]

Щоб зрозуміти за яким принципом працює веб-сайт використаємо додаткове ПО Chrome DevTools. Chrome DevTools - це набір інструментів веб-розробників, вбудованих безпосередньо в браузер Google Chrome. DevTools може допомогти вам швидко редагувати сторінки та швидко діагностувати проблеми, що в кінцевому підсумку допомагає швидше створювати веб-сайти.

Утиліта *Network panel* дозволяє переглянути усі HTTP запити які відбуваються при роботі веб-сторінки.[11]

Запис мережевих запитів - За замовчуванням *DevTools* записує всі мережеві запити на панелі "*Network*", поки *DevTools* відкритий

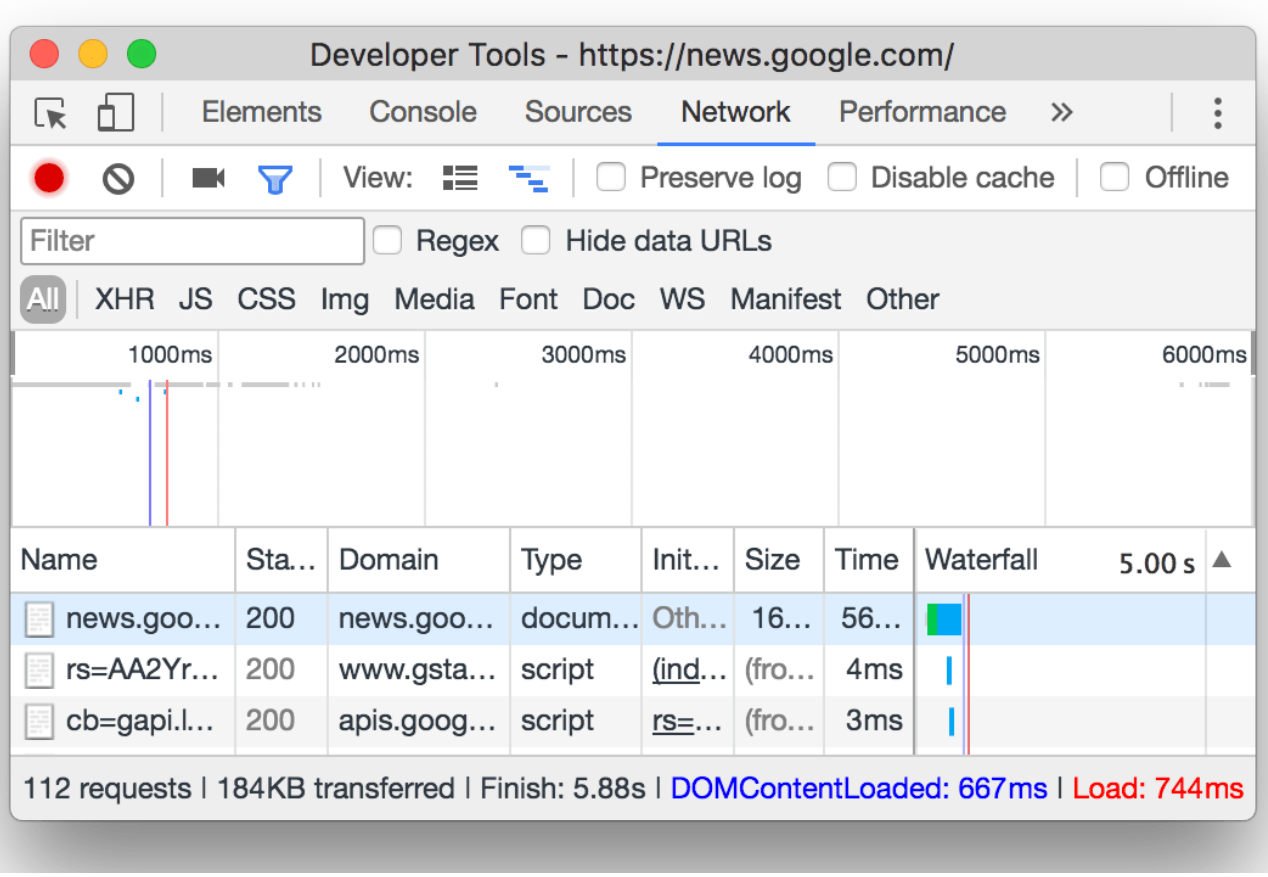

Рис 2.1 Панель *Network* у *DevTools*

Щоб зупинити запит запису:

- Клацніть Stop recording network log Зупинення мережі запису увійдіть на панель «Мережа». Стає сірим, щоб вказати, що DevTools більше не записує запити.
- Натисніть Command + E (Mac) або Control + E (Windows, Linux), коли фокусується панель Network.

Очистити запити: Клацніть *Clear* на панелі «*Network*», щоб очистити всі запити з таблиці Запити.

Фільтрування запитів: Використовуйте текстове поле Фільтр для фільтрування запитів за властивостями, такими як домен або розмір запиту.

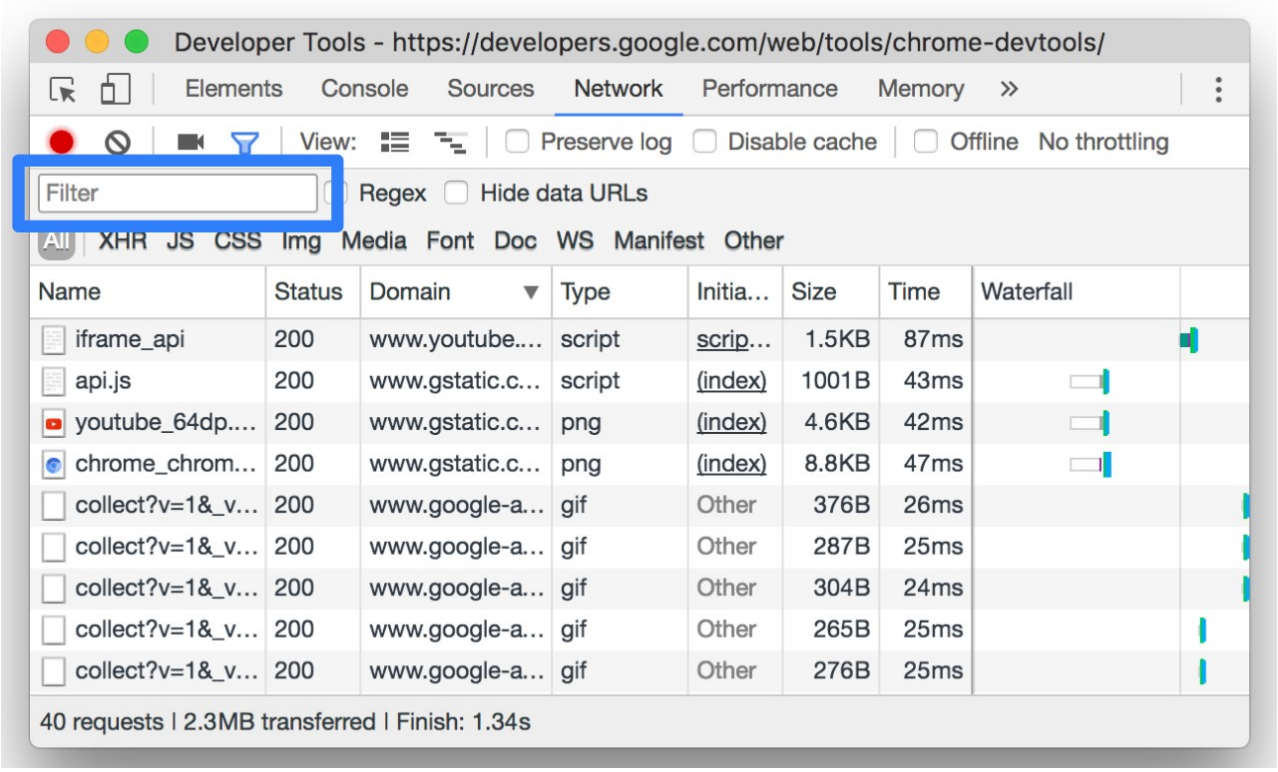

Рис 2.2 Фільтрування запитів

Проаналізуйте запити: Поки *DevTools* відкритий, він записує всі запити на панель «*Network*». Використовуйте панель "*Network*" для аналізу запитів.

Перегляд журналу запитів: Використовуйте таблицю Запити, щоб переглянути журнал усіх запитів, зроблених під час відкриття програми

DevTools. Клацання або наведення на них запитів виявляє більше інформації про них.

Ви можете використовувати кілька властивостей одночасно, розділяючи кожну властивість пробілом. Наприклад, *mime-type:image/gif larger-than:1K* відображає всі GIF-файли, що перевищують один кілобайт. Ці фільтри з декількома властивостями еквівалентні операціям AND. АБО операції наразі не підтримуються.

Нижче наведено повний перелік підтримуваних властивостей:

- *domain*: Відображати лише ресурси із зазначеного домену. Ви можете використовувати символ підстановки (*) для включення декількох доменів. Наприклад, * .com відображає ресурси з усіх доменних імен, що закінчуються на .com. *DevTools* заповнює спадне меню автозаповнення усіма доменами, з якими він стикався.

● *has-response-header:* Показати ресурси, які містять вказаний заголовок відповіді HTTP. *DevTools* заповнює спадання автозаповнення всіма заголовками відповідей, з якими він стикався.

● *is:* Використовується: працює для пошуку ресурсів *WebSocket*.

● *larger-than:* Показуйте ресурси, що перевищують вказаний розмір, у байтах. Встановлення значення *1000* еквівалентно встановленню значення 1k.

● *method:* Показати ресурси, отримані за вказаним типом методу HTTP. *DevTools* заповнює спадне меню всіма методами HTTP, з якими він стикався.

● *mime-type:* Показати ресурси вказаного типу MIME. *DevTools* заповнює спадне меню з усіма типами MIME, з якими він стикався.

● *mixed-content:* Показати всі ресурси змішаного контенту (*mixed-content:all*) або лише ті, що відображаються в даний час (*mixed-content:displayed*).

● *scheme:* Показати ресурси, отримані через незахищений HTTP (схема: http) або захищений HTTPS (схема: https).

● *set-cookie-domain:* Покажіть ресурси, які мають заголовок Set-Cookie з атрибутом Domain, який відповідає вказаному значенню. DevTools заповнює автозаповнення всіма доменами cookie, з якими він стикався.

● *set-cookie-name:* Покажіть ресурси, які мають заголовок Set-Cookie з ім'ям, яке відповідає вказаному значенню. DevTools заповнює автозаповнення усіма іменами файлів cookie, з якими він стикався.

● *set-cookie-value:* Покажіть ресурси, у яких заголовок *Set-Cookie* має значення, яке відповідає заданому значенню. *DevTools* заповнює автозаповнення усіма значеннями файлів cookie.

● *status-code:* Показуйте лише ресурси, чий код статусу HTTP відповідає вказаному коду. *DevTools* заповнює спадне меню автозаповнення всіма кодами статусу, з якими він стикався.

Фільтруйте запити за типом: щоб фільтрувати запити за типом запиту, натисніть кнопки XHR, JS, CSS, Img, Media, Font, Doc, WS (WebSocket), Manifest або Other (будь-який інший тип, не вказаний тут) на панелі Мережа.

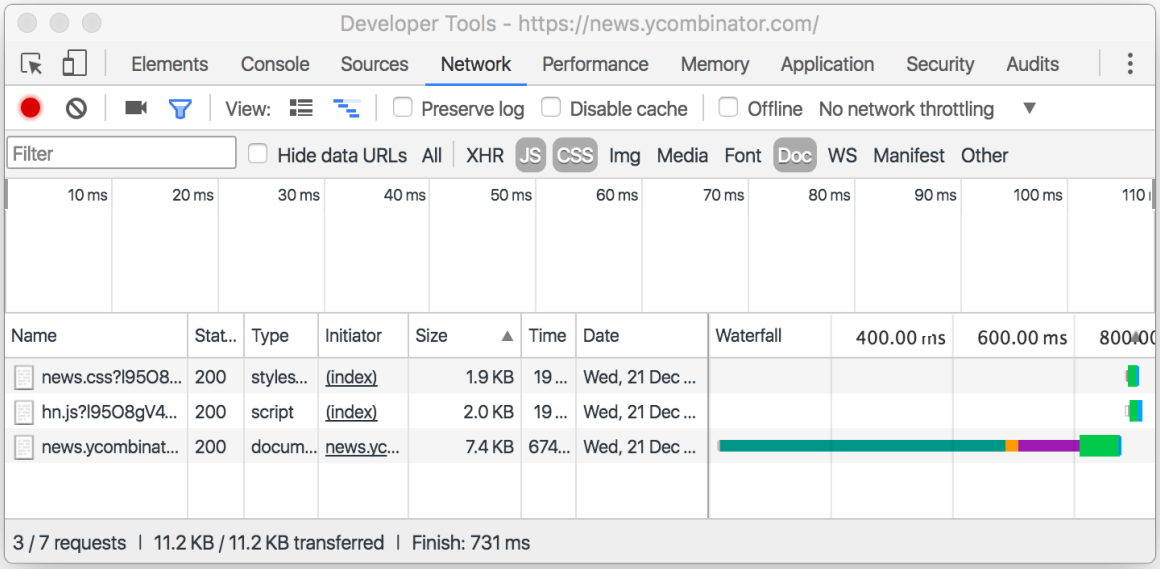

Рис 2.3 Фільтр запитів за типом

Проаналізуйте запити: поки *DevTools* відкритий, він записує всі запити на панель «*Network*». Використовуйте панель "Network" для аналізу запитів.

Перегляд журналу запитів: використовуйте таблицю Запити, щоб переглянути журнал усіх запитів, зроблених під час відкриття програми *DevTools*. Клацання або наведення на них запитів виявляє більше інформації про них.

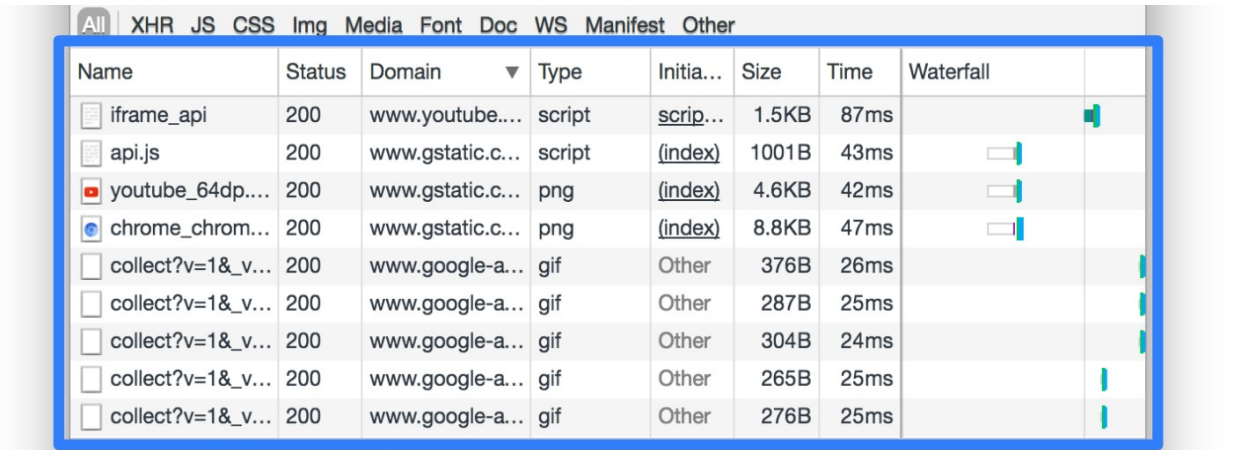

Рис 2.4 Аналіз запитів

У таблиці *Requests* за замовчуванням відображаються наступні стовпці:

- *Name* - Ім'я файла або ідентифікатор ресурсу.
- *Status* - Код статусу HTTP.
- *Type* - Тип MIME запитуваного ресурсу.
- *Initiator* - Наступні об'єкти або процеси можуть ініціювати запити:
- *Parser* - HTML-аналізатор Chrome
- *Redirect* - Перенаправлення HTTP.
- *Script* - Функція JavaScript.
- *Other* - Деякі інші дії або дії, такі як навігація до сторінки за посиланням або введення URL-адреси в адресний рядок.
- *Size* - Комбінований розмір заголовків відповідей плюс тіла відповіді, що надається сервером.
- *Time* - Загальна тривалість, від початку запиту до отримання остаточного байту у відповіді.
- *Waterfall* - Візуальний розподіл діяльності кожного запиту.

Перегляд *HTTP Headers*

Щоб переглянути дані заголовка HTTP про запит:

1) Клацніть на URL-адресу запиту під стовпцем *Name* таблиці Запити.

2) Перейдіть на вкладку *Headers*.

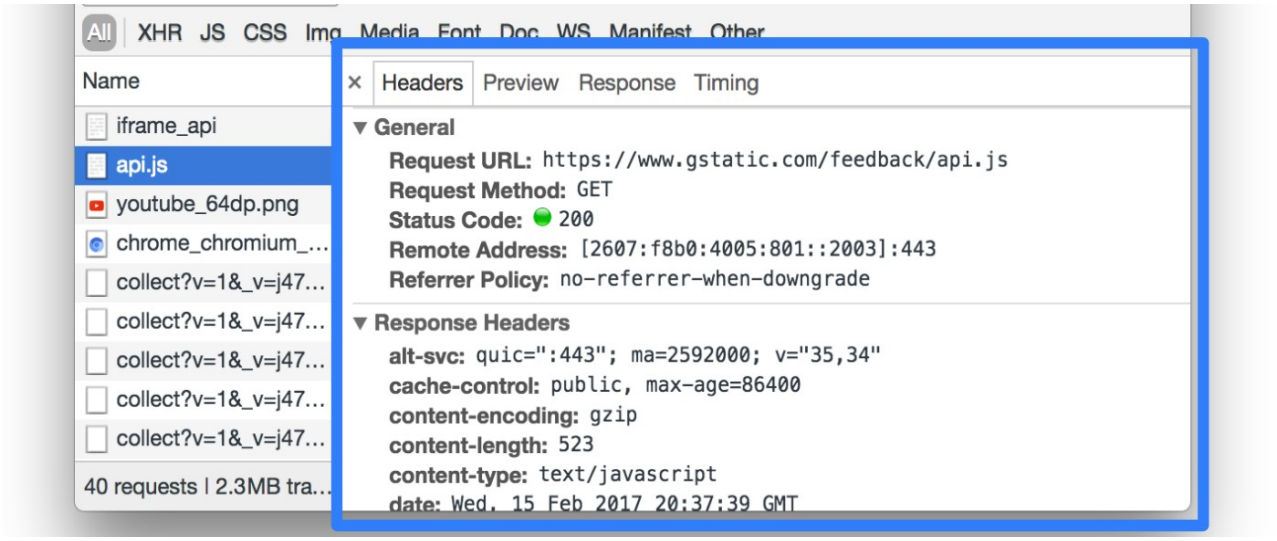

Рис 2.5 Перегляд заголовку запиту

Перегляд параметрів HTTP запиту: щоб переглянути параметри рядка запиту URL-адреси у читальному для людини форматі:

Відкрийте вкладку *Headers* для запиту, який вас цікавить.

Перейдіть до розділу *Query String Parameters*.

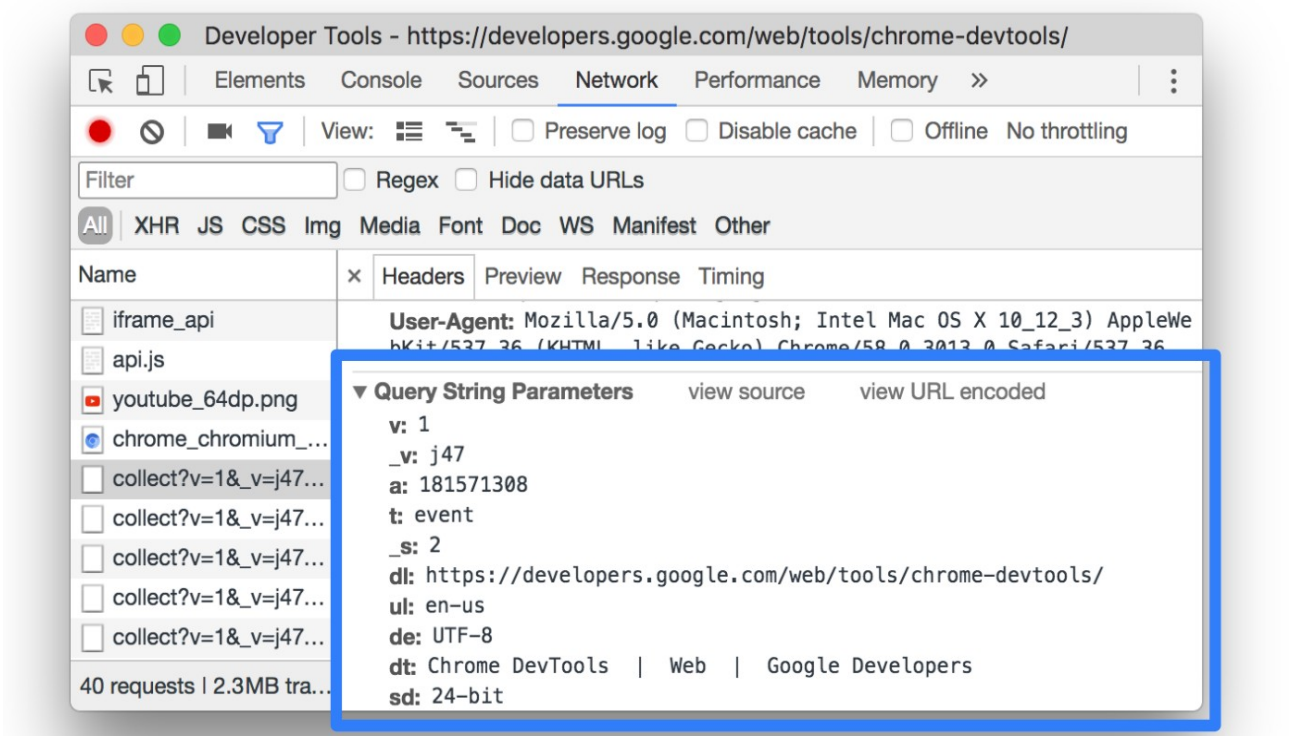

Рис 2.6 Параметри запиту HTTP

Переглянути джерело параметрів рядка запиту.

Щоб переглянути параметр рядка запиту джерела запиту:

1. Перейдіть до розділу *Query String Parameters*.
2. Клацніть на *view source*.

Переглянути параметри рядка запиту, кодовані URL-адресою

Для перегляду параметрів рядка запиту у читаному для людини форматі, але із збереженими кодуваннями:

1. Перейдіть до розділу *Query String Parameters*.
2. Клацніть *view URL encoded*.

2.5. Регулярні вирази

В програмуванні, регулярний вираз (від англ. regular expression) — це рядок, що описує або збігається з множиною рядків, відповідно до набору спеціальних синтаксичних правил. Вони використовуються в багатьох текстових редакторах та допоміжних інструментах для пошуку та зміни тексту на основі заданих шаблонів. Багато мов програмування підтримують регулярні вирази для роботи з рядками. Наприклад, *Perl* та *Tcl* мають потужний механізм для роботи, вбудований безпосередньо в їх синтаксис.

Регулярні вирази базуються на теорії автоматів та теорії формальних мов. Ці розділи теоретичної кібернетики займаються дослідженням моделей обчислення (автомати) та способами описання та класифікації формальних мов.

Регулярний вираз (часто називається шаблон) є послідовністю, що описує множину рядків. Ці послідовності використовують для точного описання множини без перелічення всіх її елементів. Наприклад, множина, що складається із слів «грати» та «ґрати» може бути описана регулярним виразом «[гґ]рати». В більшості формалізмів, якщо існує регулярний вираз, що описує задану множину, тоді існує нескінченна кількість варіантів, які описують цю множину.

Саме завдяки регулрним виразам буде проводитися парсинг даних отриманих з веб-сайту, в не залежності від їх формату, за вийнятком формату JSON так як він має чітку структуру.

Висновок до розділу 2

Щодо загальної структури проекту було вирішено використовувати скриптову мову програмування PHP з розширеннями, які дозволяють працювати з базою даних та відправляти HTTP запити, які необхідні для збору інформації. Для побудови запиту нам необхідно проаналізувати веб-сайт за допомогою *Google Chrome DevTools*, завдяки якому ми можете переглянути усі запити і обрати ті, які завантажують дані оголошення. Саме ці запити ми будемо емулювати в PHP.

РОЗДІЛ 3

Структура веб-сервісу

Автоматизація інформаційного порталу буде проводитись завдяки веб-сервісу, він буде побудований як окрема частина порталу, тому містить свій веб-сервер, в нашому випадку *NGINX*, та базу даних. Структура сервісу складається з таких компонентів:

1. ядро
2. сервіс-провайдери
3. сервіси
4. роутинг та контролери

3.1. Ядро веб-сервісу

Ядро - основна структурна модель, котра проводить завантаження усього програмного забезпечення, перевіряє залежності та запускає додаток.

```
class Core
{
   protected static $instance = null;
   protected function __construct()
   {
      self::init_environment();
      if (getenv('APP_DEBUG')) {
         ini_set('display_errors', 1);
         ini_set('display_startup_errors', 1);
         error_reporting(E_ALL);
      }
}
```

```
ini_set('max_execution_time', 1800);
        ini_set('memory_limit', '-1');
        CoreServiceProvider::instance();
    }
    public static function instance()
    {
        if (null == self::$instance) {
            self::$instance = new self;
        }
        return self::$instance;
    }
    protected static function init_environment()
    {
        $env_file = __DIR__.'/../.env';
        if (! file_exists($env_file)) {
            echo 'Could not load .env file!';
            die();
        }
        $handle = @fopen($env_file, "r");
        if ($handle) {
            while (($buffer = fgets($handle, 4096)) !== false) {
                if (strlen(trim($buffer))) {
                    putenv(trim($buffer));
                }
            }
            fclose($handle);
        }
    }
}
```

Створене ядро використовує паттерн *Singleton* - шаблон проектування, відноситься до класу твірних шаблонів. Гарантує, що клас матиме тільки один екземпляр, і забезпечує глобальну точку доступу до цього екземпляра. Під час створення екземпляру проходить перевірка файла з налаштуваннями (*.env*) та завантажується *CoreServiceProvider.*

3.2. Сервіс-провайдери

Сервіс-провайдери лежать в основі первинного завантаження всіх додатків, їх основна роль - завантажити базові функції та методи, необхідні для роботи сервісу. Для веб-сервісу необхідні такі сервіс-провайдери:

CoreServiceProvider - основний провайдер, який проводить завантаження всіх інших провайдерів.

```
class CoreServiceProvider
{
   protected static $instance = null;
   protected function __construct()
   {
      foreach (self::$services as $service) {
         if (method_exists($service, 'instance')) {
            call_user_func($service.'::instance');
         }
      }
   }
   public static function instance()
   {
      if (null == self::$instance) {
         self::$instance = new self;
```

```
        }
        return self::$instance;
    }
    static $services = [
        DB::class,
        Router::class,
        SentryServiceProvider::class
    ];
}
```

DB - провайдер перевіряє і створює з'єднання з БД, також данна модель містить сервіс, який спрощує роботу з БД, її розглянемо пізніше.

```
class DB implements ServiceProviderInterface
{
    /**
     * @var PDO
     */
    protected $db;
    protected static $instance = null;
    protected function __construct()
    {
        $db          = getenv('DB');
        $db_name     = getenv('DB_NAME');
        $db_host     = getenv('DB_HOST');
        $db_user     = getenv('DB_USER');
        $db_password = getenv('DB_PASSWORD');
       $this->db  =  new  PDO("{$db}:dbname={$db_name};host={$db_host}",
$db_user, $db_password);
```

```
    }
    public static function instance()
    {
        if (null == self::$instance) {
            self::$instance = new self;
        }
        return self::$instance;
    }
}
```

Router - провайдер по обробці HTTP запитів, він проводить аналіз запиту та запускає необхідний метод контроллеру.

```
class Router implements ServiceProviderInterface
{
    protected $get  = [];
    protected $post = [];
    protected static $instance = null;
    protected function __construct()
    {
    }
    public static function instance()
    {
        if (null == self::$instance) {
            self::$instance = new self;
        }
        return self::$instance;
    }
    public static function get($route, $action)
```

```
    {
        $instance = self::instance();
        $instance->get[$route] = $action;
    }
    public static function post($route, $action)
    {
        $instance = self::instance();
        $instance->post[$route] = $action;
    }
    public static function start()
    {
        $instance = self::instance();
$route  = substr(parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH), 1);
        $method = strtolower($_SERVER['REQUEST_METHOD']);
        if (isset($instance->$method) && is_array($instance->$method)) {
            if (array_key_exists($route, $instance->$method)) {
                $action = explode('@', $instance->$method[$route]);

                if (class_exists($action[0])) {
                    if (forward_static_call($action) === false) {
                        echo 'Action '.$action[1].' not found!';
                    }
                } else {
                    echo 'Controller '.$action[0].' not found!';
                }
            } else {
                echo 'Undefined route!';
            }
        } else {
```

```
            echo 'Undefined method '.$method.'!';
        }
    }
}
```

Розглянемо детальніше методи реалізовані в моделі сервіс-провайдеру *Router*. Тут використовується паттерн *Singleton,* для його реалізації створений метод *instance()* та властивість *$instance.* Методи *get($route, $action)* та *post($route, $action)* реєструють нові url адреси.

Метод *start()* завантажує відповідний контроллер, для цього він перевіряє чи зареєстрована url адреса у відповідному методі запиту, у випадку його відсутності повертає помилку, інакше - викликає метод класу контроллеру.

3.3. Сервіси веб-сервісу

Сервіси - моделі та методи, які спрощують роботу з додатком. Найпопулярнішими є сервіси по роботі з БД та кешем. В нашому сервісу також присутній сервіс для роботи з *CRON.*

Сервіс роботи з БД реалізує необхідні методи для роботи, при цьому спрощує роботу порівняно з роботою з *PDO,* так як для простих запитів не потрібно писати SQL запити.

```
class DB implements ServiceProviderInterface
{
    /**
     * @var PDO
     */
    protected $db;
    /**
     * @param $table
```

```
     * @param array $values
     * @return bool|string
     */
    public function insert($table, array $values)
    {
        $keys = implode(', ', array_keys($values));
        $value_keys = implode(', ', array_map(function ($key) {
            return ':'.$key;
        }, array_keys($values)));
                    $query = "INSERT INTO `{$table}` ({$keys}) VALUES
({$value_keys});";
        $stmt = $this->db->prepare($query);
        foreach ($values as $key => $value) {
            $stmt->bindValue(':'.$key, $value, PDO::PARAM_STR);
        }
        $result = $stmt->execute();
        return $result ? $this->db->lastInsertId() : false;
    }
    /**
     * @param $table
     * @param array $where
     * @param bool $order
     * @param string $order_type
     * @return array
     */
  public function select($table, array $where, $order = false, $order_type = '')
    {
        $where_data = [];
        $execute    = [];
```

```
    foreach ($where as $key => $item) {
        if (is_array($item) && array_key_exists('sign', $item)) {
            $where_data[]         = $item['key'].$item['sign'].':'.$item['key'];
            $execute[$item['key']] = $item['value'];
        } else {
            $where_data[]  = $key.'=:'.$key;
            $execute[$key] = $item;
        }
    }
    $where_data = implode(' AND ', $where_data);
    if (empty($where)) {
        $where_data = 1;
    }
    $query = "SELECT * FROM {$table} WHERE {$where_data}";
    if ($order) {
        $query .= " ORDER BY {$order} {$order_type}";
    }
    $stmt = $this->db->prepare($query);
    $stmt->execute($execute);
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
/**
 * @param $table
 * @param array $where
 * @param array $values
 * @return bool
 */
public function update($table, array $where, array $values)
{
```

```
    $where_data = [];
    $execute    = [];
    $set_data = [];
    foreach ($values as $key => $value) {
        $set_data[] = $key.'=?';
        $execute[]  = $value;
    }
    foreach ($where as $key => $item) {
        if (is_array($item) && array_key_exists('sign', $item)) {
            $where_data[] = $item['key'].$item['sign'].'?';
            $execute[]    = $item['value'];
        } else {
            $where_data[] = $key.'=?';
            $execute[]    = $item;
        }
    }
    $set_data = implode(', ', $set_data);
    $where_data = implode(' AND ', $where_data);
$query = "UPDATE {$table} SET {$set_data} WHERE {$where_data}";
    $stmt = $this->db->prepare($query);
    return $stmt->execute($execute);
}
/**
 * @param $table
 * @param array $where
 * @return bool
 */
public function delete($table, array $where)
{
```

```
        $where_data = [];
        $execute    = [];
        foreach ($where as $key => $item) {
            if (is_array($item) && array_key_exists('sign', $item)) {
                $where_data[]         = $item['key'].$item['sign'].':'.$item['key'];
                $execute[$item['key']] = $item['value'];
            } else {
                $where_data[]  = $key.'=:'.$key;
                $execute[$key] = $item;
            }
        }
        $where_data = implode(' AND ', $where_data);
        $query = "DELETE FROM {$table} WHERE {$where_data}";
        $stmt = $this->db->prepare($query);
        return $stmt->execute($execute);
    }
    /**
     * @return PDO
     */
    public function getDB()
    {
        return $this->db;
    }
}
```

Реалізований сервіс дозволяє працювати з БД не використовуючи *SQL* запити, модель описує базові необхідні функції, а саме:

insert - вносить дані у БД вказаній таблиці;

select - повертає масив значень з БД, з вказаними критеріями в конкретній таблиці;

update - оновлює дані у БД, приймає параметри - таблицю, критерії та нові значення;

delete - видаляє дані з БД, параметри - таблиця та критерії;

getDB - повертає модель *PDO*, у випадку роботи з нею на пряму.

Для виконання задач по парсингу даних за допомогою утиліти *CRON* необхідно зробити розклад та сервіс, котрий буде проводити перевірку, запускати та перевіряти задачі, це *Scheduler* сервіс. Також цей сервіс перевіряє виконувану задачу на періодичність і у разі необхідності додає у розклад нову.

```
class Scheduler
{
    const STATUS_PROCESSING = 'processing';
    const STATUS_QUEUE      = 'queue';
    const STATUS_SUCCESS    = 'success';
    const STATUS_FAILED     = 'failed';
    const STATUS_SKIPPED    = 'skipped';
    protected function __construct()
    {
    }
    public static function runSchedule($id)
    {
        $actions = (DB::instance())->select('scheduler', ['id' => $id]);
        if (empty($actions)) {
            return;
        }
```

```
    $action = $actions[0];
    if (class_exists($action['action'])) {
        $instance = new $action['action'];
        $params   = unserialize($action['params']);
        if ($instance instanceof SchedulerInterface) {
            try {
                echo 'Running scheduler: '.$action['action'].'</br>';
                $instance->run($params);
                echo 'Status: success!</br>';
            } catch (\Exception $e) {
                var_dump($e);
                echo 'Status: failed!</br>';
            }
        }
    }
}
/**
 * @param $action
 * @param $start
 * @param array $params
 * @return bool|string
 */
public static function add_schedule($action, $start, $params = [])
{
    $db = DB::instance();
    return $db->insert('scheduler', [
        'action' => $action,
        'status' => self::STATUS_QUEUE,
        'start'  => time() + $start,
```

```
            'params' => serialize($params)
        ]);
    }
    public static function checkAndRun($forced)
    {
        $db = DB::instance();
        $action = self::getCronAction($forced);
        if (class_exists($action['action'])) {
            $instance = new $action['action'];
            $params   = unserialize($action['params']);
            if ($instance instanceof SchedulerInterface) {
                try {
                        $db->update('scheduler', ['id' => $action['id']], ['status' =>
self::STATUS_PROCESSING]);
                    echo 'Running scheduler: '.$action['action'].'</br>';
                    $instance->run($params);
                        $db->update('scheduler', ['id' => $action['id']], ['status' =>
self::STATUS_SUCCESS]);
                    echo 'Status: success!</br>';
                    if ($instance->getRepeatTime()) {
                        self::add_schedule(
                            $action['action'],
                            $instance->getRepeatTime(),
                            $params
                        );
                    }
                } catch (Exception $e) {
                    var_dump($e);
```

```
                    $db->update('scheduler', ['id' => $action['id']], ['status' =>
self::STATUS_FAILED]);
                    echo 'Status: failed!</br>';
                }
            }
        } else {
                $db->update('scheduler', ['id' => $action['id']], ['status' =>
self::STATUS_FAILED]);
        }
    }
    protected static function updateOldSchedules()
    {
        $db = DB::instance();
        $db->update('scheduler', [
            'status' => self::STATUS_FAILED,
            [
                'key'   => 'start',
                'sign'  => '<',
                'value' => time() - 60 * 20
            ]
        ], ['status' => self::STATUS_QUEUE]);
    }
    /**
     * @param bool $forced
     * @return array
     */
    public static function getCronAction($forced = false)
    {
        $db = DB::instance();
```

```
    $actions = $db->select('scheduler', [
        'status' => self::STATUS_QUEUE,
        [
            'key'   => 'start',
            'sign'  => '<',
            'value' => time()
        ]
    ], 'start');
    if (empty($actions) && $forced) {
        $actions = $db->select('scheduler', [
            'status' => self::STATUS_QUEUE
        ], 'start');
    }
    if (empty($actions)) {
        echo 'No available schedules!';
        die();
    }
    return $actions[0];
}
protected static function getNextCronAction()
{
    $db = DB::instance();
    $actions = $db->select('scheduler', [
        'status' => self::STATUS_QUEUE,
        [
            'key'   => 'start',
            'sign'  => '<',
            'value' => time()
        ]
```

```
            ], 'start');
            if (empty($actions) || ! isset($actions[1])) {
                return false;
            }
            return $actions[1];
        }
        /**
         * @return array
         */
        public static function getAllSchedules()
        {
            $classes = scandir(__DIR__.'/../schedulers');
            $classes = array_filter($classes, function ($filename) {
return strpos($filename, '.php') && strpos($filename, 'Interface') === false;
            });
            return array_map(function ($filename) {
                return "Schedulers\\".str_replace('.php', '', $filename);
            }, $classes);
        }
    }
```

Сервіс реалізує всі необхідні методи, для роботи з *CRON:*

- runSchedule - запускає вказану задачу, вне залежності від часу;
- add_schedule - додає в росклад нову задачу;
- checkAndRun - перевіряє та запускає задачі за розкладом;
- updateOldSchedules - оновлення задач, котрі при виконанні отримали помилку;
- getCronAction - повертає задачу, котра повинна виконуватись в даний час, якщо вона існує;

- getNextCronAction - повертає наступну задачу;
- getAllSchedules - повертає список усіх задач які є в розкладі.

3.4. Роутинг та контролери

Роутинг - маршрутизация HTTP запитів, описує тип запиту, url адресу та метод контролеру, котрий обробляє запит. Для роботи веб-сервісу нам необхідний такий роутер:

- *\Core\Router::get('init', '\Controllers\MainController@initProject')* - даний маршрут запускає ініціалізацію сервісу, а саме - створює таблиці у БД та додає в росклад задачі;
- *\Core\Router::get('truncate/db', '\Controllers\MainController@truncate')* - очищає поточні дані з БД;
- *\Core\Router::get('cron', '\Controllers\MainController@cron')* - викликає задачу з розкладу;
- *\Core\Router::get('cron/list', '\Controllers\MainController@getSchedules')* - повертає список усіх задач у розкладі;
- *\Core\Router::get('estates', '\Controllers\EstateController@getAll')* - повертає список усіх оголошень з їх інформацією.

Висновок до розділу 3

Структура веб-сервіс повинна містити мінімальну кількість компонентів та методів, але достатню для її функціонування, для оптимізації розробки сервісу та його роботи. Функціонал ядра сервісу для автоматизації роботи порталу містить також провайдер для роботи з розкладом, щоб дані оновлювались автоматично. Простий роутинг дозволяє отримати всі необхідні дані - статус виконання задач з розкладу та отримані дані оголошень.

РОЗДІЛ 4

Розробка автоматизованого порталу нерухомості

4.1. Аналіз і створення моделі оголошення

Створення моделі оголошення має важливу роль у роботі сервісу, так як вона повинна містити всі необхідні дані о нерухомості.

Обов'язкових властивості оголошення:

- адреса;
- поштовий індекс;
- координати;
- контактна інформація - ПІБ, емейл, номер телефону та мессенджер (не обов'язковий);
- тип нерухомості;
- ціна.

Також оголошення містить не обов'язкові властивості:

- дата створення оголошення;
- площа;
- кількість кімнат;
- галерея фотографій;
- вартість комунальних послуг;
- опис;
- депозит.

Для всіх обов'язкових властивостей створені валідатори, крім координат. Координати отримуються після перевірки сервісом *Google Maps API*. Поле опис перевіряється на наявність контактної інформації в ньому.

Побудуємо модель оголошення нерухомості, а для спрощення роботи з даними згрупуємо їх.

```
class AddressEstate
    {
        private $address;
        private $postCode;
        private $coordinates;
        /**
         * AddressEstate constructor.
         * @param $address
         * @param $postCode
         * @param $coordinates
         */
        public function __construct($address, $postCode, $coordinates)
        {
            $this->address     = $address;
            $this->postCode    = $postCode;
            $this->coordinates = $coordinates;
        }
        public function save($post_id)
        {
            update_post_meta($post_id, '_iwp_address', $this->address);
            update_post_meta($post_id, '_iwp_zip', $this->postCode);
            update_post_meta($post_id, '_iwp_map', $this->coordinates);
        }
    }
```

Побудована модель *AddressEstate* містить у собі всі дані щодо адреси нерухомості.

```
class ContactsEstate
```

```
{
    private $message;
    private $name;
    private $telephone;
    private $email;
    /**
     * ContactsEstate constructor.
     * @param bool $messages
     * @param bool $name
     * @param bool $phone
     * @param bool $email
     */
    public function __construct($messages, $name, $phone, $email)
    {
        $this->message   = $messages;
        $this->name      = $name;
        $this->telephone = $phone;
        $this->email     = $email;
    }
    /**
     * @param $post_id
     */
    public function save($post_id)
    {
        delete_post_meta($post_id, '_iwp_contact_infomation');
        foreach ($this->getAttributes() as $attribute) {
            if ($this->$attribute) {
                add_post_meta($post_id, '_iwp_contact_infomation', $attribute);
            }
```

```
        }
    }
    /**
     * @return array
     */
    protected function getAttributes()
    {
        return [
            'message',
            'name',
            'telephone',
            'email'
        ];
    }
}
```

Модель *ContactsEstate* зберігає контактну інформацію про власника нерухомості.

```
class DetailsEstate
{
    private $date;
    private $area;
    private $rooms;
    private $heating;
    /**
     * DetailsEstate constructor.
     * @param $date
     * @param $area
```

```
     * @param $rooms
     * @param $heating
     */
    public function __construct($date, $area, $rooms, $heating)
    {
        $this->date    = strtotime($date) ?? time();
        $this->area    = intval($area);
        $this->rooms   = intval($rooms);
        $this->heating = intval($heating);
    }
    /**
     * @param $post_id
     */
    public function save($post_id)
    {
        update_post_meta($post_id, '_iwp_takeoverdate', $this->date);
        update_post_meta($post_id, '_iwp_area_size', $this->area);
        update_post_meta($post_id, '_iwp_bedrooms', $this->rooms);
        update_post_meta($post_id, '_iwp_heatingusage', $this->heating);
    }
    /**
     * @param $rooms
     * @return $this
     */
    public function setRooms($rooms)
    {
        $this->rooms = $rooms;
        return $this;
    }
```

```
    /**
     * @param $area
     * @return $this
     */
    public function setArea($area)
    {
        $this->area = $area;
        return $this;
    }
    /**
     * @param $heating
     * @return $this
     */
    public function setHeating($heating)
    {
        $this->heating = $heating;
        return $this;
    }
}
```

Модель *DetailsEstate* містить інформацію щодо самої нерухомості - кількість кімнат, площу та тип опалення.

```
class RentEstate
{
    private $rent_monthly;
    private $addition_cost;
    private $deposit;
    /**
```

```
     * RentEstate constructor.
     * @param float $rent_monthly
     * @param float $addition_cost
     * @param float $deposit
     */
    public function __construct($rent_monthly, $addition_cost, $deposit)
    {
        $this->rent_monthly  = $rent_monthly;
        $this->addition_cost = $addition_cost;
        $this->deposit       = $deposit;
    }
    /**
     * @param $post_id
     */
    public function save($post_id)
    {
        update_post_meta($post_id, '_iwp_price', $this->rent_monthly);
        update_post_meta($post_id, '_iwp_utilities', $this->addition_cost);
        update_post_meta($post_id, '_iwp_deposit', $this->deposit);
    }
     /**
     * @param $post_id
     * @return float
     */
    public static function getTotalRent($post_id)
    {
$addition = get_post_meta($post_id, '_iwp_utilities', true)['meta_value'];
        return self::getMonthlyPrice($post_id) + $addition;
    }
```

```
        /**
         * @param $post_id
         * @return float
         */
        public static function getMonthlyPrice($post_id)
        {
            return get_post_meta($post_id, '_iwp_price', true)['meta_value'];
        }
    }
```

Модель *RentEstate* містить інформацію о вартості нерухомості.

```
class Estate
{
    private $post_id;
    private $title;
    private $description;
    /**
     * @var AddressEstate
     */
    private $address;
    /**
     * @var GalleryEstate
     */
    private $gallery;
    private $crawl_id;
    private $crawl_site;
    /**
     * @var ContactsEstate
```

```
 */
private $contacts;
/**
 * @var DetailsEstate
 */
private $details;
/**
 * @var RentEstate
 */
private $rent;
/**
 * @var TermEstate
 */
private $terms;
/**
 * Estate constructor.
 * @param $post_id
 * @param $title
 * @param $description
 * @param AddressEstate $address
 * @param GalleryEstate $gallery
 * @param ContactsEstate $contacts
 * @param DetailsEstate $details
 * @param RentEstate $rent
 * @param TermEstate $terms
 * @param $crawl_id
 * @param $crawl_site
 */
public function __construct(
```

```
    $post_id,
    $title,
    $description,
    AddressEstate $address,
    GalleryEstate $gallery,
    ContactsEstate $contacts,
    DetailsEstate $details,
    RentEstate $rent,
    TermEstate $terms,
    $crawl_id,
    $crawl_site
) {
    $this->post_id    = $post_id;
    $this->title      = ucfirst(strtolower(str_replace(['*', '!'], '', $title)));
    $this->address    = $address;
    $this->gallery    = $gallery;
    $this->crawl_id   = $crawl_id;
    $this->crawl_site = $crawl_site;
    $this->contacts   = $contacts;
    $this->details    = $details;
    $this->rent       = $rent;
    $this->terms      = $terms;
    $desc = $description;
    $desc = CrawlHelper::replaceEmail($desc);
    $desc = CrawlHelper::replacePhone($desc);
    $this->description = $desc;
}
public function save($meta = true)
{
```

```
    if ($this->isInvalid()) {
        return false;
    }
    if (! $this->post_id) {
        $this->post_id = self::insert_estate(
            $this->title,
            $this->description,
            'publish',
            $this->crawl_id,
            $this->crawl_site
        );
    }
    if ($meta) {
        $this->update_meta();
    }
}
protected function update_meta()
{
    $this->contacts->save($this->post_id);
    $this->address->save($this->post_id);
    $this->details->save($this->post_id);
    $this->gallery->save($this->post_id);
    $this->terms->save($this->post_id);
    $this->rent->save($this->post_id);
}
/**
 * @param $title
 * @param $desc
 * @param $status
```

```
     * @param $crawl_id
     * @param $crawl_class
     * @return bool
     */
    public static function insert_estate($title, $desc, $status, $crawl_id, $crawl_class)
    {
        $db = DB::instance();
        return $db->insert('estates', [
            'title'       => $title,
            'description' => $desc,
            'status'      => $status,
            'crawl_id'    => $crawl_id,
            'crawl_class' => $crawl_class,
            'ready'       => 0
        ]);
    }
    /**
     * @return bool
     */
    public function isInvalid()
    {
        return $this->terms->isInvalid() || $this->rent->isInvalid() || $this->details->isInvalid();
    }

    public static function delete($id)
    {
        $db = DB::instance();
        $db->delete('estates', ['id' => $id]);
```

```
        $db->delete('attachments', ['estate_id' => $id]);
        $db->delete('estates_meta', ['estate_id' => $id]);
        $db->delete('estates_term', ['estate_id' => $id]);
    }
}
```

Основна модель оголошення, яка містить у собі всі попередні моделі. Модель містить додаткові методи, для спрощення роботи з оголошенням:

- *save* - перевіряє на валідність оголошення, створює його, за необхідністю та зберігає додаткову інформацію з моделей;
- *update_meta* - зберігає додаткову інформацію з моделей;
- *insert_estate* - створює нове оголошення у БД;
- *delete* - видаляє оголошення і додаткову інформацію.

4.2. Модель сайту по збору даних

Модель сайту, з якого буде проводитись збір даних буде складатись з двох класів, перший відповідає за пагінацію та отримання списку оголошень, другие - за дані оголошення.

Розробимо абстрактну модель для пагінації.

```
abstract class BasePaginator
{
    /**
     * @param int $page
     * @return array
     */
    public function getEstates($page = 1)
    {
        $estates = [];
```

```
    while (true) {
        $current = $this->getEstateFrom($this->getPage($page));
        if (empty($current)) {
            break;
        }
        $estates = array_merge($estates, $current);
        $page++;
    }
    return $estates;
}
/**
 * @param $html
 * @return array
 */
protected function getEstateFrom($html)
{
    $result = [];
    $ids    = $this->getObjectID($html);
    $prices = $this->getPrices($html);
    if (empty($ids) || count($prices) !== count($ids)) {
        return [];
    }
    for ($i = 0; $i < count($ids); $i++) {
        $result[] = (object)[
            'id'    => $ids[$i],
            'price' => $prices[$i]
        ];
    }
    return $result;
```

```
    }
    /**
     * @param $page
     * @return string
     */
    protected function getPage($page)
    {
        $result = false;
        while (! $result) {
            $result = $this->getHtml($page, CrawlHelper::getProxyService());
        }
        return $result;
    }
    /**
     * @param $page
     * @param ProxyInterface|null $proxyService
     * @return bool|string
     */
    abstract public function getHtml($page, ProxyInterface $proxyService = null);
    /**
     * @param $html
     * @return array
     */
    abstract protected function getObjectID($html);
    /**
     * @param $html
     * @return array
     */
    abstract protected function getPrices($html);
```

}

Модель складається з декількох реалізованих методів та абстрактних, розглянемо їх детальніше:

getEstates - метод, який повертає масив оголошень, складається з циклу який перебирає всі сторінки на веб-сайті;

getEstateFrom - метод, який проводить злиття двох масивів(цін та індикаторів);

getPage - метод, приймає номер сторінки в якості параметру, повертає відповідь від веб-сайту, вне залежності від формату;

getHtml - абстрактний метод, повертає відповідь від веб-сайту;

getObjectID - абстрактний метод, повертає список індикаторів;

getPrices - абстрактний метод, повертає список цін.

Розробимо модель конкретної сторінки оголошення.

```
abstract class BaseWebsite
{
   abstract public function getEstateHtml($crawl_id);
   abstract public function addEstate($estate);
   abstract protected function getUrl();
   abstract public static function getEstateAddress($html): AddressEstate;
 abstract public static function getEstateGallery(array $images): GalleryEstate;
   abstract public static function getEstateRent($estate): RentEstate;
   abstract public static function getEstateDetails($estate): DetailsEstate;
   abstract public static function getEstateDescription($estate_id);
   abstract public static function getEstateTerms($estate): TermEstate;
}
```

Модель складається лише з абстрактних методів, які потрібно буде реалізувати, а саме:

- *getEstateHtml* - повертає HTML структуру сторінки оголошення;
- *addEstate* - основний метод, проводить збір усіх даних та зберігає їх;
- *getUrl* - повертає URL адресу сторінки оголошення;
- *getEstateAddress* - проводить парсинг сторінки та повертає модель *AddressEstate*;
- *getEstateGallery* - проводить парсинг сторінки та повертає масив зі списку зображень;
- *getEstateRent* - проводить парсинг сторінки та повертає модель *getEstateRent*;
- *getEstateDetails* - проводить парсинг сторінки та повертає модель *getEstateDetails*;
- *getEstateTerms* - проводить парсинг сторінки та повертає модель *getEstateTerms*;
- *getEstateDescription* - повертає опис оголошення.

Реалізуємо один із веб-сайтів, наприклад https://wohnen.tag-ag.com

```
class TagWohnenPaginator extends BasePaginator
{
    /**
     * @param $page
     * @param ProxyInterface|null $proxyService
     * @return bool|string
     */
    public function getHtml($page, ProxyInterface $proxyService = null)
    {
```

```
        $ch = curl_init();
 curl_setopt($ch, CURLOPT_URL, $this->getUrl().'?'.$this->getBody($page));
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        if ($proxyService) {
          curl_setopt($ch, CURLOPT_PROXY, $proxyService->getProxyString());
curl_setopt($ch, CURLOPT_PROXYTYPE, $proxyService->getCurlProxyType());
        }
        $result = curl_exec($ch);
        if (curl_error($ch) !== "") {
            error_log(curl_error($ch));
            return false;
        };
        curl_close($ch);
        return $result;
    }
    /**
     * @param $html
     * @return array
     */
    protected function getObjectID($html)
    {
        if ($c = preg_match_all("/.*?(\")(detail_)(\\d+)(\")/is", $html, $matches)) {
            return $matches[3];
        }
        return [];
    }
    /**
     * @param $html
     * @return array|mixed
```

```
     */
    protected function getPrices($html)
    {
                                                if      ($c      =      preg_match_all("/.*?(<dt
class=\"odd\">).*?(\\d+)(,).*?(EUR)(<\/dt>)/is", $html, $matches)) {
            return $matches[2];
        }
        return [];
    }
    /**
     * @return string
     */
    protected function getUrl()
    {
      return 'https://wohnen.tag-ag.com/no_cache/tag-wohnen-immobiliensuche/';
    }
    /**
     * @param $page
     * @return string
     */
    public static function getBody($page)
    {
        return http_build_query([
            'tx_hmimmoscoutconnector_immoscoutsearch' => [
                'page'      => $page,
                'newFilter' => [
                    'estateType'       => 'Wohnung',
                    'commercialization' => 'Miete',
                    'city'             => '',
```

```
            ]
        ]
    ]);
  }
}
```

Метод *getHtml* виконує HTTP запит емулюючи дані звичайного браузеру, підставляючи поле *tx_hmimmoscoutconnector_immoscoutsearch* отпимане завдяки утиліті *Google Chrome DevTools*.

Регулярні вирази, реалізовані у методах *getObjectID* та *getPrices* проводять пошук шаблону по сторінці. Переглянемо HTML структуру отриманої сторінки.

```
<div class="singleResult odd filterCategoryApartmentRent cityFilter">
<span class="cityData Mnchengladbach" style="display:none;">
Mönchengladbach</span>
<a ref="/no_cache/tag-wohnen-immobiliensuche/expose/immobilie/84109104/"
class="singleResultLink" target="detail_84109104">
<span class="imageColumn">
<span class="imageContainer">
<img src="https://pictures.immobilienscout24.de/listings/bc432929-1b32-464b-
a5e3-31a1d0969435-958414046.jpg/ORIG/resize/60x60%3E/extent/60x60/format/
jpg/quality/50" alt="Vorschaubild" title="Wohnen">
</span>
</span>
<span class="headlineColumn">
<span class="estateType">Einziehen, sich wohlfühlen! Gartennutzung!</span>
</span>
<span class="middleColumn">
```

Mönchengladbach / Mülfort

Mülgaustr. 279

Garten

**

**

<dl>

<dd class="even">Wohnfläche</dd>

<dt class="even"> 52,00 m²</dt>

<dd class="odd">Anzahl Räume</dd>

<dt class="odd"> 1 </dt>

<dd class="odd"> Warmmiete</dd>

<dt class="odd"> 400,00 EUR</dt>

</dl>

**

* *

**

* *

Ідентифікатор оголошення прихований у атрибут *target* і приймає значення з маскою *detail_* (*target="detail_84109104"*). Тому регулярний вираз побудований з приміткою на маску.

Значення вартості оренди знаходиться в *<dt class=”odd”>* але для відокремлення від іншого параметру додамо конкретизацію у вигляді валюти - *EUR.*

class TagWohnen extends BaseWebsite

{

const PREFIX = 'tagwohnen';

*/***

** @var ProxyService*

```
*/
private $proxyService;
/**
* Wohnraumkarte constructor.
* @param ProxyInterface|null $proxy
*/
public function __construct(ProxyInterface $proxy = null)
{
$this->proxyService = $proxy;
}
/**
* @param $crawl_id
* @return bool|string
* @throws Exception
*/
public function getEstateHtml($crawl_id)
{
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $this->getUrl().$crawl_id.'/');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
if ($this->proxyService) {
curl_setopt($ch, CURLOPT_PROXY, $this->proxyService->getProxyString());
curl_setopt($ch, CURLOPT_PROXYTYPE, $this->proxyService-
>getCurlProxyType());
}
$result = curl_exec($ch);
if (curl_error($ch) !== "") {
throw new Exception(curl_error($ch));
};
```

```
curl_close($ch);
return $result;
}
/**
* @param $estate
* @return bool|Estate|null
*/
public function addEstate($estate)
{
if (CrawlHelper::isEstateExist($estate->id, self::PREFIX)) {
return true;
}
try {
$html = $this->getEstateHtml($estate->id);
} catch (Exception $e) {
error_log($e->getMessage(), null, $e->getTraceAsString(), $e->getFile());
return null;
}
if (empty($html)) {
return null;
}
$address     = self::getEstateAddress($html);
$gallery     = self::getImages($html);
$rent        = self::getEstateRent($html);
$details     = self::getEstateDetails($html);
$description = self::getEstateDescription($html);
$terms       = self::getEstateTerms($html);
$contacts    = new ContactsEstate(true, false, false, false);
$entity = new Estate(
```

```
false,
self::getTitle($html),
$description,
$address,
$gallery,
$contacts,
$details,
$rent,
$terms,
$estate->id,
self::PREFIX);
$entity->save();
return $entity;
}
/**
* @return string
*/
protected function getUrl()
{
return 'https://wohnen.tag-ag.com/no_cache/tag-wohnen-immobiliensuche/expose/immobilie/';
}
/**
* @param $html
* @return AddressEstate
*/
public static function getEstateAddress($html): AddressEstate
{
```

```
$address = trim(str_replace("'", '', CrawlHelper::getJSValue($html, 'mapsAddress')));
$zip = explode(' ', explode(', ', $address)[1])[0];
$lat = trim(CrawlHelper::getJSValue($html, 'mapsLatitude'));
$lon = trim(CrawlHelper::getJSValue($html, 'mapsLongitude'));
if ($lat && $lon) {
$coord = implode(',', [$lat, $lon]);
} else {
$coord = '';
}
return new AddressEstate($address, $zip, $coord);
}
/**
* @param array $images
* @return GalleryEstate
* @deprecated
*/
public static function getEstateGallery(array $images): GalleryEstate
{
return new GalleryEstate();
}
/**
* @param $html
* @return RentEstate
*/
public static function getEstateRent($html): RentEstate
{
$monthly = self::toFloat(self::getTableValue($html, 'Kaltmiete'));
$addition = self::toFloat(self::getTableValue($html, 'Nebenkosten'));
```

```
$deposit  = self::toFloat(self::getTableValue($html, 'Kaution'));

return new RentEstate($monthly, $addition, $deposit);
}
/**
* @param $html
* @return DetailsEstate
*/
public static function getEstateDetails($html): DetailsEstate
{
$area    = self::getTableSTR($html, 'Größe:');
$rooms   = intval(self::getTableSTR($html, 'Art'));
$heating = self::getTableSTR($html, 'Energiekennwert:');
return new DetailsEstate(date('d.m.Y'), $area, $rooms, $heating);
}
/**
* @param $html
* @param $key
* @return string
*/
protected static function getTableSTR($html, $key)
{
$start = strpos($html, $key);
if (! $start) {
return '';
}
$start = strpos($html, '">', $start);
if (! $start) {
return '';
```

```
}
$start += 2;
$end   = strpos($html, '</td>', $start);
return substr($html, $start, ($end - $start));
}
/**
* @param $html
* @return string
*/
public static function getEstateDescription($html)
{
return self::getDescriptionBlock($html,
'furnishingDescription').self::getDescriptionBlock($html,
'longDescription');
}
/**
* @param $html
* @return TermEstate
*/
public static function getEstateTerms($html): TermEstate
{
$terms = new TermEstate();
$terms->add('iwp_features', self::getFeatures($html));
$terms->add('iwp_heatingtype', self::getTableSTR($html, 'Heizung:'));
$terms->add('iwp_type', 'Wohnung');
$terms->add('iwp_status', 'rent');
return $terms;
}
/**
```

```
* @param $html
* @return array
*/
public static function getFeatures($html)
{
$features_str = CrawlHelper::getContentByAttribute($html, 'class="innerTable paddingLeft"');
if (empty($features_str)) {
return null;
}
$features_str = trim(strip_tags($features_str[0]));
return explode(', ', $features_str);
}
/**
* @param $html
* @return GalleryEstate
*/
public static function getImages($html)
{
if (preg_match_all("/.*?(<img).*?(src=\").*?(pictures\\.immobilienscout24\\.de)(.*?)(.jpg).*?(alt=\")(.*?)(\")/is",
$html, $matches)) {
$gallery = new GalleryEstate();
foreach ($matches[4] as $i => $path) {
$url = 'https://pictures.immobilienscout24.de'.$path.'.jpg';
$gallery->addImage($url, $matches[7][$i], '');
}
return $gallery;
}
```

```
return new GalleryEstate();
}
/**
* @param $html
* @param $key
* @return string|null
*/
public static function getTableValue($html, $key)
{
if (preg_match_all("/.*?(<td).*?(>)({$key})(<\/td>).*?(>)(.*?)(<\/td>)/is", $html, $matches)) {
return $matches[6][0];
}
return null;
}
/**
* @param $val
* @return float
*/
protected static function toFloat($val)
{
return (float)str_replace(',', '.', str_replace('.', '', $val));
}
/**
* @param $html
* @param $id
* @return string
*/
protected static function getDescriptionBlock($html, $id)
```

```
{
$start = strpos($html, $id);
if (! $start) {
return '';
}
$start = strpos($html, '<td>', $start);
if (! $start) {
return '';
}
$start += 4;
$end   = strpos($html, '</td>', $start);
return trim(substr($html, $start, ($end - $start)));
}
}
```

Метод *getEstateHtml* побудований аналогічно методу пагинатора, за відмінністю у відсутності параметрів для запиту.

Методи для створення моделі даних (*getEstateAddress, getEstateGallery, getEstateRent, getEstateDetails, getEstateTerms)* формують дані за допомогою регулярних виразів.

4.3. Модель задачі з розкладу

Автоматизація оновлення даних проводиться завдяки утиліті *CRON*, яка буде запускати скрипт сервісу *SchedulerServiceProvider*. Скрипт перевіряє на наявність нові задачі і запускає їх. Розглянемо абстрактну модель задачі по обробці оголошень з сайту і реалізуємо її.

```
abstract class EntitySchedule implements SchedulerInterface
{
   /**
    * @param array $params
```

```
     * @return bool
     */
    abstract public function run($params = []);
    /**
     * @return int|bool
     */
    public function getRepeatTime()
    {
        return false;
    }
    /**
     * @param $entities
     * @param $entity
     * @return bool
     */
    protected function isRepeat($entities, $entity)
    {
        if (count($entities) !== 1) {
            return false;
        }
        return ((array)$entities[0])['id'] === ((array)$entity)['id'];
    }
}
```

Абстрактна модель містить додаткові методи, які необхідні для створення періодичності повторення даної задачі.

Реалізуємо модель задачі по отриманню списку оголошень з веб-сайту https://wohnen.tag-ag.com:

```
class TagWohnenSchedule implements SchedulerInterface
```

```
{
const PIECES_SIZE = 15;
/**
* @param array $params
* @return bool
*/
public function run($params = [])
{
try {
$site        = new TagWohnenPaginator();
$list        = $site->getEstates();
$new_estates = [];
$old = CrawlHelper::getListToDrafting($list, TagWohnen::PREFIX);
if (! empty($old)) {
CrawlHelper::draftList($old);
}
foreach ($list as $estate) {
$id = CrawlHelper::isEstateExist($estate->id, TagWohnen::PREFIX);
if (! $id) {
$new_estates[] = $estate;
}
}
$pieces = array_chunk($new_estates, self::PIECES_SIZE);
$delay  = 180;
foreach ($pieces as $piece) {
Scheduler::add_schedule(
GlobalEntitySchedule::class,
$delay,
['piece' => $piece, 'prefix' => TagWohnen::PREFIX]
```

```
);
$delay += 360;
}
} catch (Exception $e) {
error_log($e->getMessage(), null, $e->getTraceAsString(), $e->getFile());
}
return true;
}
/**
* @return int|bool
*/
public function getRepeatTime()
{
return 60 * 60 * 24;
}
}
```

Створена задача є основною для обробки даного веб-сайту, але під час виконання вона створює дочірні задачі, по отриманню даних з конкретних оголошень(нових) і виставляє їм затримку, щоб не перевантажувати сервер. В дочірню задачу передається масив з 15 оголошень для подальшої обробки та аналізу, розглянемо його:

```
class TagWohnenEntitySchedule implements SchedulerInterface
{
/**
* @param array $params
* @return bool
*/
public function run($params = [])
```

```
{
foreach ($params['piece'] as $entity) {
$proxy      = CrawlHelper::getProxyService();
$crawl_site = new TagWohnen($proxy);
$estate     = $crawl_site->addEstate((object)$entity);
if ($estate === null && ! $this->isRepeat($params['piece'], $entity)) {
Scheduler::add_schedule(TagWohnenEntitySchedule::class, 360, ['piece' =>
[$entity]]);
}
unset($crawl_site);
unset($class);
}
return false;
}

/**
* @return int|bool
*/
public function getRepeatTime()
{
return false;
}
/**
* @param $entities
* @param $entity
* @return bool
*/
protected function isRepeat($entities, $entity)
{
```

```
if (count($entities) !== 1) {
return false;
}
return $entities[0]['id'] === $entity['id'];
}
}
```

Задача *TagWohnenEntitySchedule* виконує по чергову обробку оголошень та у разі їх валідності зберігає у БД. У разі виникнення помилки під час обробки задача створює свою копію.

4.4. Інтеграція сервісу з інформаційним порталом

Інтеграція сервісу буде проводитись по протоколу HTTP для цього створимо метод контролеру, котрий буде повертати актуальний список оголошень у форматі JSON.

```
class EstateController
{
public static function getAll()
{
$db = DB::instance()->getDB();
$result = $db->query('SELECT * FROM estates WHERE ready = 1')->fetchAll(\
PDO::FETCH_ASSOC);
$meta = $db->query('SELECT estate_id,meta_key,meta_value FROM
estates_meta')->fetchAll(\PDO::FETCH_GROUP | \PDO::FETCH_ASSOC);
$term = $db->query('SELECT estate_id,term,taxonomy FROM estates_term')-
>fetchAll(\PDO::FETCH_GROUP | \PDO::FETCH_ASSOC);
$attachments = $db->query('SELECT estate_id,title,description,url FROM
attachments')->fetchAll(\PDO::FETCH_GROUP | \PDO::FETCH_ASSOC);
```

```
foreach ($result as &$item) {
$item_id = $item['id'];
$item['meta'] = self::get_data($meta, $item_id);
$item['term'] = self::get_data($term, $item_id);
if (strcmp($item['crawl_class'], 'gwh') !== 0) {
$item['attachment'] = self::get_data($attachments, $item_id);
} else {
$item['attachment'] = [];
}
unset($meta[$item_id]);
unset($term[$item_id]);
unset($attachments[$item_id]);
}
header('Content-Type: application/json');
echo json_encode($result);
return true;
}
/**
* @param $data
* @param $item_id
* @return array
*/
protected static function get_data($data, $item_id)
{
return array_key_exists($item_id, $data) ? $data[$item_id] : [];
}
}
```

При запиті на сервер з URL */estates* отримуємо у відповідь JSON з масивом оголошень, які містять всю необхідну інформацію. Тому на стороні

інформаційного порталу необхідно зробити логіку відправки запиту та оновлення даних.

```
class Scheduler
{
const HOOK        = 'crawling_global_schedule';
const HOOK_IMAGES = 'crawling_image_schedule';
const PIECES_SIZE = 15;
function __construct()
{
add_filter('cron_schedules', __CLASS__.'::add_schedules');
add_action(self::HOOK, __CLASS__.'::process_global_schedule');
add_action(self::HOOK_IMAGES, __CLASS__.'::process_images_schedule');
add_action('before_delete_post', __CLASS__.'::remove_gallery', 999);
}
protected static $instance = null;
/**
* Return an instance of this class.
* @since     1.0.0
*
*/
public static function instance()
{
if (null == self::$instance) {
self::$instance = new self;
}
return self::$instance;
}
public static function addScheduleEvent()
```

```
{
wp_clear_scheduled_hook(Scheduler::HOOK);
wp_schedule_event(time() + 60 * 15, 'crawling_time', Scheduler::HOOK);
}
/**
* @param array $schedules
* @return array
*/
public static function add_schedules(array $schedules)
{
$period = get_option(PREFIX.'_scheduler_period');
if (! $period) {
$period = 4;
}
$schedules['crawling_time'] = [
'display'  => $period.' day(days)',
'interval' => DAY_IN_SECONDS * doubleval($period)
];
return $schedules;
}
public static function process_global_schedule()
{
$entities = CrawlService::getDataFromApi();
self::removeOldEstates($entities);
foreach ($entities as $estate) {
$id  = CrawlHelper::isEstateExist($estate->crawl_id, $estate->crawl_class);
$new = false;
if ($id && get_post_meta($id, '_crawl_desync', true) === 'on') {
continue;
```

```
}
if (! $id) {
$new = true;
$id = wp_insert_post([
'post_title'   => $estate->title,
'post_content' => $estate->description,
'post_author'  => 1,
'post_type'    => 'iwp_property',
'post_status'  => 'pending'
]);
update_post_meta($id, '_crawl_id', $estate->crawl_id);
update_post_meta($id, '_crawl_class', $estate->crawl_class);
update_post_meta($id, '_iwp_featured', true);
}
update_post_meta($id, '_iwp_property_id', $id);
if (! empty($estate->attachment)) {
self::createSingleSchedule($id, self::HOOK_IMAGES);
}
$eTerm = new TermEstate();
foreach ($estate->term as $term) {
$eTerm->add($term->taxonomy, unserialize($term->term));
}
$eTerm->save($id);
foreach ($estate->meta as $meta) {
update_post_meta($id, $meta->meta_key, $meta->meta_value);
}
wp_update_post([
'ID'           => $id,
'post_title'   => $estate->title,
```

```
'post_content' => $estate->description,
'post_status'  => $new ? 'pending' : $estate->status
]);
}
}
/**
* @param array $list
*/
public static function removeOldEstates($list)
{
$old = self::getListToDrafting($list);
foreach ($old as $item) {
if (get_post_meta($item->post_id, '_crawl_desync', true) === 'on'){
continue;
}
wp_delete_post($item->post_id, true);
}
}
/**
* @param array $list
* @param string $class
* @return array
*/
public static function getListToDrafting($list, $class = false)
{
$list = implode(', ', array_map(function ($item) {
return "'{$item->crawl_id}'";
}, $list));
global $wpdb;
```

```
if ($class) {
$result = $wpdb->get_results("SELECT id.post_id FROM `{$wpdb->prefix}postmeta` AS class
INNER JOIN `{$wpdb->prefix}postmeta` AS id ON id.post_id = class.post_id
WHERE class.meta_key = '_crawl_class' AND class.meta_value = '{$class}'
AND id.meta_key = '_crawl_id' AND id.meta_value NOT IN ({$list})");
} else {
$result = $wpdb->get_results("SELECT id.post_id FROM `{$wpdb->prefix}postmeta` AS class
INNER JOIN `{$wpdb->prefix}postmeta` AS id ON id.post_id = class.post_id
WHERE class.meta_key = '_crawl_class'
AND id.meta_key = '_crawl_id' AND id.meta_value NOT IN ({$list})");
}
return $result;
}
/**
* @param $post_id
*/
public static function process_images_schedule($post_id)
{
if (get_post_status($post_id) === false) {
var_dump($post_id.'post_not found');
return;
}
$attachments = get_post_meta($post_id, '_crawling_attachments', true);
if (! $attachments) {
var_dump($post_id.'no attachments');
return;
}
```

```
$gallery = new GalleryEstate();
if ($attachments) {
foreach ($attachments as $img) {
$gallery->addImage($img->url, $img->title, $img->description);
}
}
$gallery->save($post_id);
wp_update_post([
'ID'          => $post_id,
'post_status' => 'publish'
]);
}
/**
* @param $post_id
* @param $hook
*/
protected static function createSingleSchedule($post_id, $hook)
{
$delay = rand(10, 60);
while (wp_schedule_single_event(time() + $delay, $hook, compact('post_id')) ===
false) {
$delay += $delay;
}
}
public static function remove_gallery($post_id)
{
global $post_type;
if ($post_type !== 'iwp_property') {
return;
```

```
}
$thumbnail   = get_post_meta($post_id, '_thumbnail_id');
$cover_image = get_post_meta($post_id, '_iwp_cover_image');
$gallery     = get_post_meta($post_id, '_thumbnail_id');
foreach (array_merge($thumbnail, $cover_image, $gallery) as $img) {
wp_delete_attachment($img, true);
}
return;
}
}
```

Логіка оновлення даних розбита на два методи(для оптимізації роботи):

- *process_global_schedule* - метод проводить перевірку та вносить дані у БД порталу і створює нові задачі *process_images_schedule*;
- *process_images_schedule* - метод завантажує зображення та прикріплює їх до оголошення.

Висновок до розділу

Розроблений сервіс зі збору даних оголошень повністю автоматизований та дозволяє розробнику легко масштабувати кількість оголошень та кількість сайтів з яких проводиться збір даних. Завдяки реалізованим інтерфейсам додавання нового веб-сайту просте та оптимальне. Розклад задач дозволяє проводити масштабування сервісу без погіршення його продуктивності. Так як сервіс відокремлений від порталу то при роботі з сервісом не потрібно модифікувати портал і при цьому оголошення завжди залишаються актуальними.

ВИСНОВКИ

В дипломній роботі було розроблено автоматизований сервіс, який проводить аналіз та збір оголошень, з різних веб-сайтів. При цьому перевіряє оголошення на відповідні критерій та валідує їх. Завдяки розподіленній структурі система легко масштабується як у випадку з кількістю оголошень, так і кількість веб-сайтів. Функціонування сервісу повністю автоматизовано завдяки розробленій системі розкладу та утиліті CRON.

Інформаційний портал нерухомості відокремлений від сервісу та розроблений з використанням CMS WordPress. Завдяки можливості підключати плагіни портал інтегрується з сервісом та проводить синхронізацію автоматично, так як WordPress має базові включають в себе розклад задач. Тому для повного функціонування порталу нерухомості не потрібна наявність модераторів,

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BF%D0%BE%D1%80%D1%82%D0%B0%D0%BB
2. https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D1%81%D0%B0%D0%B9%D1%82
3. https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0
4. https://uk.wikipedia.org/wiki/HTTP
5. https://uk.wikipedia.org/wiki/WebSocket
6. https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7
7. https://www.php.net/manual/ru/book.curl.php
8. https://www.php.net/manual/ru/book.pdo.php
9. https://developers.google.com/places/web-service/search
10. https://uk.wikipedia.org/wiki/AJAX
11. https://developers.google.com/web/tools/chrome-devtools